National Defence / Défense nationale

HOPPING SYNTHESIZER CONTROLLER

by

Robin Addison
EHF SATCOM Section
Electronics Division

DEFENCE RESEARCH ESTABLISHMENT OTTAWA
TECHNICAL NOTE 90-17

PCN
D6470

August 1990
Ottawa

91-03505/

Abstract

The EHF SATCOM (Extremely High Frequency Satellite Communications) project includes a contract in industry for a ground-based model of a frequency hopping spread spectrum satellite communications system. There is also an in-house effort in key technical areas. One requirement for is a Hopping Synthesizer Controller to drive a frequency synthesizer, and is the subject of this report. The controller was built on a single board using a digital signal processor for calculations and an erasable programmable logic device for interfacing. It provides the pseudo-random hopping sequence, modulation, channel selection and doppler precorrection. The Hopping Synthesizer Controller can drive the frequency synthesizer at hop rates up to 25 khops/s. Four configurations are supported: operation within either the payload or ground terminal simulator, for uplink or downlink. Each configuration supports commands from a Synchronization Controller and a special synchronization mode. In this document the software and hardware details are provided along with a user guide for the board.

Resumé

Le projet EHF SATCOM inclut un contrat donné à l'industrie pour le dévelopment d'un modèle expérimental d'un système de communications par satellite à fréquence extrêmement haute avec étalement de spectre fait par bond de fréquences. La section SATCOM travaille sur certains aspects difficiles du projet comme la synchronisation. Pour ce travail, un contrôleur de synthétiseur de fréquences est requis pour chaque liaison, ascendante et descendante, ainsi qu'à chaque terminaison de ces liaisons. Ce contrôleur, construit sur une plaquette, est composé d'une unité de traitement de signaux numériques, d'une composante logique programmable et effaçable pour l'interface, et de composantes supportées. La modulation, la sélection du canal, le nombre pseudo-aléatoire pour choisir le bond ainsi que la correction doppler sont tous utilisés pour le calcul des fréquences. Le contrôleur a un mode spécial pour la synchronisation et peut être commandé par le contrôleur de synchronisation. Le contrôleur de synthétiseur de fréquences peut fonctionner jusqu'à 25 milles bonds de fréquence par seconde. Les détails du circuit et du logiciel sont inclus dans ce rapport ainsi qu'un guide d'utilisation.

Executive Summary

For the Department of National Defence EHF SATCOM (Extremely High Frequency Satellite Communications) project, industrial contractors are building a ground based advanced development model of an EHF frequency hopped satellite communications system. To support this contract, the Defence Research Establishment Ottawa SATCOM section is working on some of the critical areas such as synchronization. Any work in this area requires a frequency hopping synthesizer such as a Comstron FS2000A, for each end of the link, that is under control of the Synchronization Controller. The Comstron synthesizer accepts a frequency word from an external port and switches to this frequency. This report describes a Hopping Synthesizer Controller which generates this word at the appropriate time and also responds to commands from the Synchronization Controller.

The design of this Hopping Synthesizer Controller includes a digital signal processor to do the calculations and respond to external commands from the Synchronization Controller. The calculations involve producing the random hop frequency, adding Frequency Shift Keying (FSK) modulation, channel selection, precorrecting the doppler offsets and frequency word format conversion for the Comstron synthesizer. The board supports the Synchronization Controller by precomputing the synchronization hop frequencies and rapidly switching them on demand. The Synchronization Controller can override any of the default parameters of the frequency calculation. The board is configurable for use in uplink or downlink mode and for either ground terminal or satellite operation.

Flexibility was a key aspect of the board design. This one board can be used in any of the four locations that require a synthesizer. The use of a digital signal processor and programmable read-only memories permit simple modification of software algorithms such as random number generation, frequency calculations or synthesizer frequency word formatting. The use of an erasable programmable logic device (EPLD) allows simple and rapid modification of the hardware interface.

The Hopping Synthesizer Controller was implemented with the new generation EPLDs that featured more complex and denser building blocks. These devices were found to be very useful in replacing the numerous small integrated circuits normally required for any design. The design and programming are done on a personal computer thus simplifying alterations within the EPLD. Substituting one EPLD for all the usual support integrated circuits meant rapid development of the hardware as well as easy customization of the hardware interface which simplified the software requirement.

A Hopping Synthesizer Controller was designed and built to drive a Comstron synthesizer in a frequency hopped communications system. This was accomplished with the board successfully tested. Future plans include replication of the board under contract.

Table of Contents

Abstract iii

Resumé iii

Executive Summary v

Table of Contents vii

1. **Introduction** 1
   - 1.1 **The Problem** 1
   - 1.3 **Hopping Synthesizer Controller Board** 1
   - 1.4 **Report Outline** 3

2. **Software** 3
   - 2.1 **General** 3
   - 2.2 **Program Structure** 3
   - 2.3 **Command Processing** 4
   - 2.4 **Algorithms** 5
     - 2.4.1 Frequency Computations 5
     - 2.4.2 BCD Conversion 7
     - 2.4.3 Random Number Generation 7

3. **Hardware** 10
   - 3.1 **General** 10
   - 3.2 **Board Construction** 10
   - 3.3 **TMS320C25 Digital Signal Processor** 12
   - 3.4 **Erasable Programmable Logic Devices** 13
   - 3.5 **Timing Consideration** 14

4. **Conclusions** 15
   - 4.1 **Summary** 15
   - 4.2 **Future Work** 15

References 17

List of Abbreviations 18

Appendix A: Schematics A1
   - 1. **General** A1
   - 2. **EPLD Schematics** A1

Appendix B: Parts List and Board Layout B1
1. Layout Description B1
2. Parts List B2

Appendix C: Synchronization Controller Interface C1
1. General C1
2. Hopping Synthesizer Controller Commands C1
3. Calculation Parameter Values C4
4. Hardware Interface C6

Appendix D: FSK/Channel Interface D1
1. General D1
2. Hardware Interface D1

Appendix E: Frequency Synthesizer Interface E1
1. General E1
2. Comstron Synthesizer E1
3. Comstron Cable E1
4. Hardware Interface E1

Appendix F: User's Guide F1
1. General F1
2. Installation F1
3. Configuration F1
4. Hopping Synthesizer Controller Modes of Operation F2
5. Hopping Synthesizer Controller Initialization F3
6. Reading Values F4
7. Synchronization F5
8. Setting a CW Frequency F6

Appendix G: Software Listings G1
1. General G1
2. Main Program (MAIN.ASM) G1
3. Calculation Subroutines (CALC.ASM) G14
4. Random Number Generation Routines (RAN.ASM) G20

1. Introduction

1.1 The Problem

For the Department of Nation Defence (DND) Extremely High Frequency Satellite Communications (EHF SATCOM) project, industrial contractors are building a ground based advanced development model of an EHF frequency hopped satellite communications system. To support this contract, the Defence Research Establishment Ottawa (DREO) Satellite Communications (SATCOM) section is working on some of the critical areas such as synchronization. Any work in this area requires a frequency hopping synthesizer, for each end of the link, that is under control of the Synchronization Controller. The Comstron FS2000 synthesizer takes a frequency word from an external port and switches to this frequency within 1 μs. This report describes a Hopping Synthesizer Controller which generates the frequency word at the appropriate time and responds to the commands of the Synchronization Controller.

1.2 Hopping Synthesizer Controller Within a System

Fig. 1 shows a simplified ground terminal block diagram using two Hopping Synthesizer Controllers. The transmit user data is collected by the Frame Multiplexer and passed to the Hopping Synthesizer Controller as necessary. The transmit Hopping Synthesizer Controller drives the Hopping Synthesizer to produce the modulated and hopped signal (including doppler precorrection) which is then upconverted and transmitted to the satellite. The signal received from the satellite downlink is downconverted, with the downlink Hopping Synthesizer Controller causing the Hopping Synthesizer to dehop the signal as well as to postcorrect for doppler offsets. The signal is then demodulated and demultiplexed to the various users. In both the uplink and downlink chains, the Synchronization Controller commands the Hopping Synthesizer Controllers. The Synchronization Controller also provides the doppler information obtained from the Ephemeris Processor. A payload diagram would be similar, but there would be no requirement for synchronization or ephemeris.

1.3 Hopping Synthesizer Controller Board

The original design for a Hopping Synthesizer Controller did not involve a microprocessor. The first board built was found to be too inflexible and did not properly support the Synchronization Controller. Based on the information learned, the second design included a digital signal processor to do the calculations and respond to external commands. This is the implementation described in this report. The calculations involve producing a random hop frequency, adding Frequency Shift Keying (FSK) modulation and channel selection, precorrecting the doppler offsets, and converting the frequency word from binary to BCD (binary coded decimal) for the Comstron synthesizer. The Synchronization Controller uses a scheme which requires rapid changing of the hop frequency during the detection of synchronization hops. The board supports the Synchronization Controller by precomputing the synchronization hop frequencies and rapidly switching them on demand. The Synchronization Controller can override any of the default parameters of the frequency calculation. The board is configurable for use in uplink or downlink mode and for either ground terminal or satellite operation.

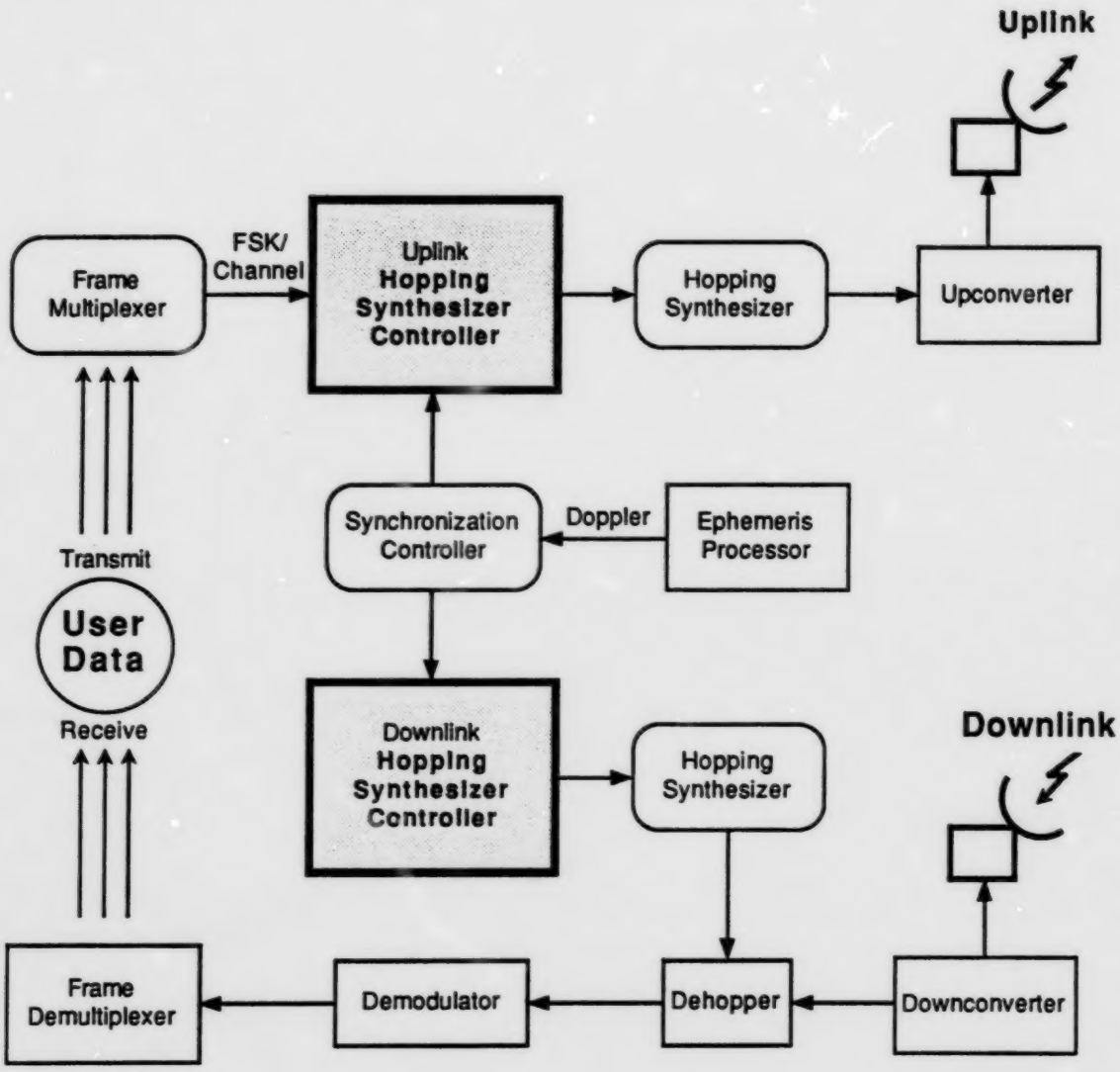


Fig. 1. Ground terminal simplified block diagram.

1.4 Report Outline

The reports consists of two major sections, software and hardware, as well as several appendices containing detailed technical information. The software section covers the structure of the digital signal processor software, the command processing sequence for Synchronization Controller commands, and the algorithms used. These algorithms cover the frequency calculations, binary to BCD conversion to format the word for the Comstron synthesizer, and the random number generation for the hop frequency pattern.

The hardware section covers the board description, the details of the digital signal processor, and the erasable programmable logic device (EPLD) used for interfacing and the steps taken to bypass a timing problem.

The appendices include the schematics, parts list, the three external interface descriptions, a user's guide and the software listings.

2. Software

2.1 General

This section could equally well be called firmware because all software is programmed into an erasable programmable read-only memory (EPROM) for the board. The EPLD is programmed as well, but since it replaces interface logic chips, the EPLD details can be found in the hardware section. All software listings and associated command files are given in Appendix G.

The Hopping Synthesizer Controller operates in one of three modes: Stop Mode where hopping is stopped, Run Mode where normal hopping takes place, and Sync Mode where precomputed hops are invoked by the Synchronization Controller.

The most commonly used commands are the Load commands to change the calculation parameters and the Change Mode commands to switch between modes. All calculation parameters can be restored to their default settings by the Set Defaults command. The Go commands are used in Sync Mode to change frequencies rapidly. The remaining commands are used for debugging: Read to allow the Synchronization Controller to examine the Hopping Synthesizer Controller's memory, RXF and SXF to control the latch timing of the hopping synthesizer.

In the following sections the software structure, Synchronization Controller command processing and the calculation algorithms are presented.

2.2 Program Structure

The software consists of three files: MAIN, CALC and RAN. The MAIN file includes the main program and subroutines for command processing (INSTR), setting default values (SETDEF) and miscellaneous short support routines. CALC contains the calculation subroutines for frequency computations (CALC), the binary to BCD conversion (BCD32 and BCDINI) and a 32 x 32 to 64-bit signed integer multiply (MUL32). The random number generation routines can be found in RAN including the random number routine (RAN and RANINI) and a routine

to step ahead in time (RANJMP). The detailed algorithms (frequency calculations, BCD conversion and random number generation) can be found in section 2.4.

2.3 Command Processing

The effectiveness of the Hopping Synthesizer Controller hinges on rapid response in Sync Mode. This is because of the synchronization scheme that the board was designed to support. This scheme involves the detection of a burst of four synchronization (sync) hops with predetermined modulation and data. Each sync hop is separated by a blank hop during which the detection computations for the previous hop occur. If a sync hop is detected, the next hops in the burst are used for confirmation. Once the detection of a burst of four sync hops is detected, another confirmation involving all four sync hops of the next burst is required. The first hop of the second burst is precomputed whereas for the final three hops the Hopping Synthesizer Controller is switched into normal Run Mode. A false alarm is deemed to have occurred if the initial sync hop is detected and the subsequent sync hops are not detected.

In Sync Mode, the Hopping Synthesizer Controller is able to respond to commands and switch frequencies in 2 μs. To allow such a rapid response it is necessary that the command decoding be very simple. This is accomplished by reducing the number of different commands to a minimum and by choosing opcodes that can be easily decoded.

The three Change Mode commands (Stop, Run, Sync) can be easily distinguished because the Change Mode Stop command has a value of 0, the Change Mode Run is negative (value 8000h = -32768) and Change Mode Sync is positive (value 4000h = 16384).

Within Sync Mode, the Go commands request the rapid change of frequency. There are five possible Go commands corresponding to the five precomputed frequencies. To simplify command decoding, the Go command opcode is identical to the address of the precomputed frequency. The Go command is loaded into an index register and then the BCD frequency that it points to is sent to the synthesizer.

To summarize Sync Mode command processing, first the processor tests for Change Mode Stop (zero flag set means switch to Stop Mode), then tests for Change Mode Run (sign bit set implies change to Run Mode) and otherwise uses the opcode as the address of the frequency word to send to the synthesizer. There is no valid opcode or range checking, so if an illegal opcode (including other commands, non-commands and Change Mode Sync) is received then indeterminate results could occur. Response time is more important than robustness (with respect to illegal commands) while in Sync Mode.

The Load commands are similar to the Go commands in that their value is the address to be used. They can be distinguished from the Go commands because they use different areas of memory (Go commands start at 60h and Load commands start at 300h). Load commands are followed by the low word and then high word of the value to be loaded.

The Read command was added strictly for debugging. The lower 12 bits of the read command specify the memory location to be read. The Read command is of the form 2mmmh where mmm is the address of the memory location to be read. The remaining commands, Set Defaults, RXF and SXF, were assigned distinct values of 0400h, 0500h and 0501h respectively.

In Stop Mode, the processor waits only for commands and processes them once all the words of the command have been received. For example, all three words of a Load command are buffered before the command is processed. In Stop Mode only legal commands are

processed and if illegal opcodes are received they are ignored.

While in Run Mode, the commands are processed only once per hop. After the new frequency is computed and if all the words of the command are received then one command will be processed. After that, the processor waits for the next hop. Invalid opcodes are ignored when received in Run Mode.

2.4 Algorithms

There are three algorithms used within the Hopping Synthesizer Controller. The first and most important one is the Frequency Computation algorithm which determines how the random number, modulation, channel number and configuration affect the desired frequency. The second algorithm formats this frequency for the Comstron synthesizer using a binary to BCD conversion. The last algorithm given is a pseudo-random number generator used to determine the hop location within the hopping bandwidth.

2.4.1 Frequency Computations

All frequency computations are performed in the routine CALC according to the following equations. The different board configuration options (uplink or downlink, ground terminal or payload) are handled by different values of the arguments. A table of these values can be found at the end of the equations.

In the calculations shown below, certain values such as $SCALE_{PN}$ and f_{base} change depending on the board configuration selected. For illustrative purposes, the values used will be those associated with the uplink from a ground terminal. This is the most complex case since it includes modulation. All the other cases are subsets of this case.

First, the available hopping bandwidth will be calculated to produce the scale value for the random number that provides the random hop frequency. Then the frequency offset associated with modulation and channel selection will be computed. These will be summed with the fixed frequency offsets at which point doppler is added. Finally the value needed for the synthesizer will be derived.

The available hopping bandwidth is 2 GHz. If a 500 kHz guard band for doppler is left at both the lower and upper edges, there is only 1.999 GHz remaining. A multichannel system could have a bandwidth as large as 1 MHz, giving a useable hopping bandwidth of 1.998 GHz.

The placement of the hop within the hopping bandwidth is determined by a 24-bit pseudorandom number (*PN*) that is generated for each hop. This provides a random number from 0 to 16777215 which is multiplied by $SCALE_{PN}$ (9990000 = $1.190900802_6 \times 2^{23}$) and then scaled by 2^{-23}. The resultant, f_{PN}, is the frequency offset of the hop from the lower edge of the hopping bandwidth. f_{PN} is in units of 100 Hz and given by

$$f_{PN} = \frac{PN \times SCALE_{PN}}{2^{23}} \qquad (1)$$

The frequency offset associated with FSK modulation and channel spacing, f_{mod}, is derived from the 8-bit FSK/Channel input. The lower 3 bits, m_{FSK}, select one of the 8-ary tones. The channel selection, m_{chan}, is from the upper 5 bits. The channels are 8 tones wide

with no interchannel spacing. The intertone spacing, Δf_{tone}, is 400, which (in units of 100 Hz) corresponds to a spacing of 40 kHz. f_{mod} is given by

$$f_{mod} = \Delta f_{tone} \times (m_{FSK} + 8 \times m_{chan}) \quad (2)$$

Neglecting, for the moment, doppler frequency offset, the nominal transmit frequency, f_{nom}, would be the sum of the low edge of the hopping bandwidth f_{base}, the lower edge guard offset to allow for negative dopplers f_{off}, the hop frequency and the modulation. f_{nom} is

$$f_{nom} = f_{base} + f_{off} + f_{PN} + f_{mod} \quad (3)$$

The actual transmitted frequency must be precorrected for doppler frequency offset so it appears as f_{nom} to the satellite. The transmit frequency, f_{tx}, is the nominal transmit frequency plus the doppler frequency. This doppler is the product of the nominal frequency and the doppler factor γ. A positive doppler factor is used here when the satellite is moving away from the ground terminal thus compensating for a negative doppler shift. On the receive side, the doppler factor sign would be reversed for postcorrection of doppler shift. For calculation accuracy, γ is stored scaled by 2^{32}. f_{tx} is given by

$$f_{tx} = f_{nom} + \frac{f_{nom} \times \gamma}{2^{32}} \quad (4)$$

The Comstron synthesizer has a frequency range of 0-4 GHz. The frequency desired from the hopping synthesizer, f_{synth}, is the transmitted frequency less the upconverter local oscillator (LO) frequency f_{LO}.

$$f_{synth} = f_{tx} - f_{LO} \quad (5)$$

The default values (corresponding to an f_{synth} of 0.5-2.5 GHz on the uplink and 1-2 GHz on the downlink) for the key elements are given in the Table 1 below:

Table 1. Default calculation parameters.

Location	$SCALE_{PN}$	Δf_{tone}	f_{base}	f_{off}	γ	f_{LO}
GT Uplink	9990000 (1.998 GHz)	400 (40 kHz)	435000000 (43.5 GHz)	5000 (500 kHz)	(as per orbit x 2^{32})	430000000 (43 GHz)
GT Downlink	4977500 (0.9955 GHz)	0 (no mod)	202000000 (20.2 GHz)	2500 (250 kHz)	(as per orbit x 2^{32})	192000000 (19.2 GHz)
Payload Uplink	9990000 (1.998 GHz)	0 (no mod)	435000000 (43.5 GHz)	5000 (500 kHz)	0 (no doppler)	430000000 (43 GHz)
Payload Downlink	4977500 (0.9955 GHz)	0 (no mod)	202000000 (20.2 GHz)	2500 (250 kHz)	0 (no doppler)	192000000 (19.2 GHz)

2.4.2 BCD Conversion

The resultant frequency, f_{synth}, is the desired binary frequency for the synthesizer in units of 100 Hz. Unfortunately, the Comstron synthesizer requires a BCD (Binary Coded Decimal) command frequency. The conversion from binary to a 32-bit BCD word is accomplished in the routine BCD32. Another synthesizer, such as the Scitek synthesizer being used by Communications Research Centre (CRC), would require a different format and would use a routine other than BCD32.

The binary to BCD conversion algorithm involves repeated binary subtractions similar to binary long division. Long division uses the same dividend (though shifted) whereas the BCD conversion uses a different dividend for each decade. This dividend is 8 times the decade of conversion and would be shifted to the most significant position (it would be $8 \times 10 \times 16^M$ for the tens decade where M depends on the size of the conversion). The subtractions are in fact conditional subtractions - if the subtraction would leave a positive result then it is performed, producing a 1 for the conversion, otherwise there is no subtraction giving a 0. The TMS320C25 has a conditional subtraction instruction, but it only works with a 16-bit dividend. A 32-bit algorithm equivalent to this instruction is used for the high 16 bits, and then the conditional subtraction is used for the low 16 bits.

The binary number to be converted is given by 'value' and the BCD result is given by 'answer'. If there are N decades to be converted, the algorithm is:

```
answer = 0
do i = 0 .. N-1
   dividend = 8 x 10^(N-i-1) x 16^i
   do j = 0 .. 3
      if dividend ≤ value then
         value = value - dividend
         value = value x 2
         answer = answer x 2 + 1
      else
         value = value x 2
         answer = answer x 2 + 0
      endif
   enddo
enddo
```

The following example of a 3 decade conversion illustrates the algorithm:

Value to be converted = 279 = 117h
100s decade → dividend = $8 \times 10^2 \times 16^0$ = 800 = 320h
Answer so far = 0

1st operation: conditional subtraction of value from dividend, cannot subtract, shift answer left and add new answer bit of 0,

shift value left

Value to be converted = 558 = 22Eh
Dividend = 800 = 320h
Answer so far = 0

2nd operation: conditional subtraction, cannot subtract,
shift answer left and add new answer bit of 0,
shift value left

Value to be converted = 1116 = 45Ch
Dividend = 800 = 320h
Answer so far = 0

3rd operation: conditional subtraction, can do subtraction,
shift answer left and add new answer bit of 1,
shift value left

Value to be converted = 632 = 278h
Dividend = 800 = 320h
Answer so far = 1

4th operation: conditional subtraction, cannot subtract,
shift answer left and add new answer bit of 0,
shift value left

Value to be converted = 1264 = 4F0h
Dividend = 800 = 320h
Answer so far = 2

At this point four bits required for the 100's decade have been computed. So now change to the ten's decade, thus changing the dividend to $8 \times 10^1 \times 16^1 = 1280$.

Value to be converted = 1264 = 4F0h
Dividend = 1280 = 500h
Answer so far = 2

5th operation: cannot subtract, gives 0, answer 4
6th operation: can subtract, gives 1, answer 9
7th operation: can subtract, gives 1, answer 19 = 13h
8th operation: can subtract, gives 1, answer 39 = 27h

Now on to the one's decade giving a dividend of $8 \times 10^0 \times 16^2 = 2048$.

Value to be converted = 2304 = 900h

Dividend = 2048 = 800h
Answer so far = 39 = 27h

9th operation:	can subtract, gives 1, answer 79 = 4Fh
10th operation:	cannot subtract, gives 0, answer 158 = 9Eh
11th operation:	cannot subtract, gives 0, answer 316 = 13Ch
12th operation:	can subtract, gives 1, answer 633 = 279h

Value to be converted = 0
Answer = 633 = 279h

As can be seen by the above, the conversion is complete giving the BCD output required. The hexadecimal value of the answer is the BCD form of the original value.

The conditional subtraction instruction on the TMS320C25 has the additional feature that the answer is shifted into the low bit of the value. Thus only one 32-bit accumulator is needed and both shift operations can be done with one shift of the accumulator.The accumulator starts with the input value and ends up with the BCD output value.

To speed up conversion, the table of dividends is loaded into random access memory (RAM) by the routine BCDINI. This routine must be invoked once at the beginning of the main program, prior to any calls to the conversion routine BCD32.

2.4.3 Random Number Generation

In an operational system, the hop frequency will be chosen using an encrypted sequence. For test purposes, it is impractical to acquire and use an encryption device so a simple random number generator was chosen. [1] gives a simple implementation of a 31-bit multiplicative linear congruential random number generator. The basic equation is given below where s_{n+1} is the new random number (and next seed), s_n is the current seed and a is the multiplier. (For the hop sequence the value of $a = 16807$ was chosen because it is known to produce a long sequence). The random number is

$$s_{n+1} = as_n \bmod(2^{31} - 1) \tag{6}$$

The product as will be between 32 and 62 bits long. This product can be split into two numbers: q for the lower 31 bits and p for the upper bits then the two components are 31 bits or less and given by

$$as = 2^{31}p + q \tag{7}$$

Depending whether the sum $(p + q)$ overflows or not, (6) can be expressed as (note that the modulo operation has been avoided)

$$s_{n+1} = \begin{cases} p + q & ; p + q < 2^{31} \\ p + q - 2^{31} + 1 & ; p + q \geq 2^{31} \end{cases} \quad (8)$$

This can be thought of as discarding the overflow and incrementing the result when an overflow occurs. Equivalently, adding the overflow to the result gives

$$s_{n+1} = p + q + \text{overflow} \quad (9)$$

Equations (7) and (9) result in compact code for 31-bit random number generation. The routine RAN computes this number using these equations. The linear congruential random number generators cannot use a seed of zero, so prior to any random number generation, RANINI should be called to initialize the seed to 1.

Occasionally, it is necessary to set the clock ahead in time. The routine RANJMP rapidly steps through the random numbers until the one appropriate for the specified time-of-day is computed. There are no known means of changing time rapidly with these generators so this brute force method must be employed.

The synchronization testing will require that the sequence of random numbers be of some finite length (so one does not have to wait forever in the case of an error in the synchronization methods). For simplicity, a length less than 2^{16} was chosen giving a sequence length of 61440. It was important that this length be an integral number of sync hop burst periods long. The time incrementer automatically resets the seed to 1 at the end of the sequence.

3. Hardware

3.1 General

The block diagram of the controller is shown in Fig. 2. The Hopping Synthesizer Controller consists of a digital signal processor with program memory to do the computations and respond to commands, a latch to store the next frequency to be sent to the synthesizer, and a custom interface that latches sync processor commands, performs handshaking, generates control signals and provides the status of the configuration DIP (dual in-line package) switches. The FSK/Channel data input is only used when the Hopping Synthesizer Controller is configured for ground terminal uplink.

3.2 Board Construction

The Hopping Synthesizer Controller is driven by a TMS320C25 signal processor that performs the computations and responds to commands from the Synchronization Controller. The program is stored in 2 x 7C261 8k-byte fast EPROMS. To buffer and latch the outputs to the frequency synthesizer, 4 x 74F374 octal latches are used. 5 x 74LS244 are used to buffer offboard signal and data lines. The EPM5128 EPLD provides all the interface logic: hop clock generation from 40 MHz clock, configuration control, Synchronization Controller interface (latch and handshaking), TMS320C25 interface (port select, status and data ports), synthesizer strobe

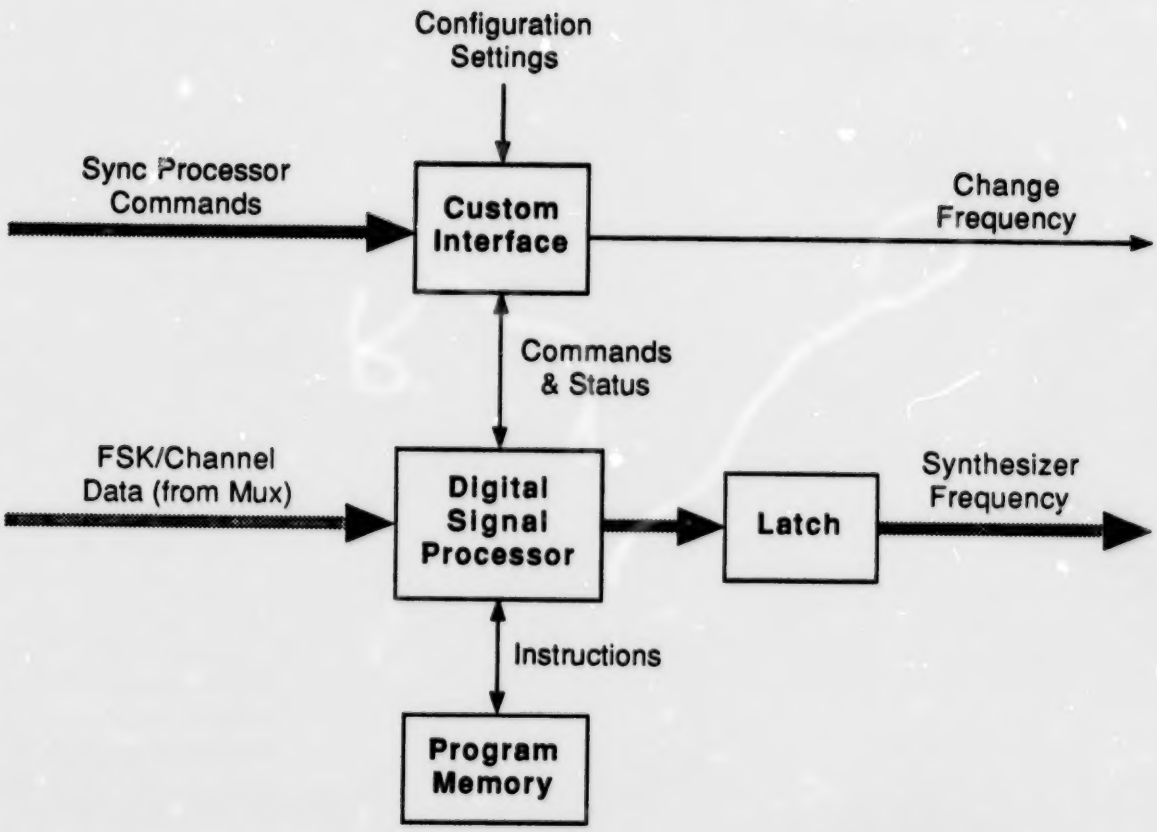


Fig. 2. Hopping Synthesizer Controller block diagram.

and onboard latch strobes. Finally, a 74LS221 dual monostable multivibrator is used for power-up reset and a crystal oscillator is used for the 40 MHz clock. This circuit was built on a 10 cm x 16 cm Speedwire prototyping board. The board schematic is given in Fig. A1 of Appendix A. The parts list can be found in Appendix B.

3.3 TMS320C25 Digital Signal Processor

The TMS320C25 (C25 for short) is the brain of the Hopping Synthesizer Controller. This digital signal processor has a good architecture for the frequency computations and is sufficiently flexible to perform the other duties. The component used was a 40 MHz part in a plastic leadless chip carrier (PLCC) package. The C25 has two modes of operation: microprocessor mode which uses external read-only memory (ROM) and microcomputer mode which uses internal and external ROM. On this board, the C25 was operated in microprocessor mode so all program memory was in an external EPROM. There is sufficient on-chip RAM for all data storage so there is no need for external RAM. This microprocessor has separate program and data storage of 64k 16-bit words each. The TMS320C25 also has an Input/Output (I/O) space with 16 read/write ports. On this board, only the lower two bits are decoded, resulting in 3 extra shadow groups. Of the four decoded ports, two are unused and are available for future expansion. The remaining two ports have different uses in read and write. The associated memory and I/O maps are given in Tables 2, 3 and 4.

Table 2. Program memory map.

Program Address	Program Memory Use
0000-1FFF	Program EPROM
2000-3FFF	Shadow* of EPROM
4000-5FFF	Shadow of EPROM
6000-7FFF	Shadow of EPROM
8000-FFFF	Shadows of EPROM

* a shadow occurs when the address upper bits are not decoded so accesses to all memory addresses that match the lower decoded bits, access the same location.

Table 3. Data memory map.

Data Address	Data Memory Use
0000-0005	On-Chip Registers
0006-005F	Reserved
0060-007F	On-Chip RAM
0080-01FF	Reserved
0200-03FF	On-Chip RAM
0400-FFFF	External (unused)

Table 4. I/O port map

Port	Read	Write
0	Command Register	Synthesizer Latch Low
1	Status Register	Synthesizer Latch High
2	Unused	Unused
3	Unused	Unused
4-7	Shadows of 0-3	Shadows of 0-3
8-11	Shadows of 0-3	Shadows of 0-3
12-15	Shadows of 0-3	Shadows of 0-3

3.4 Erasable Programmable Logic Devices

EPLDs are integrated circuits that consist of many macrocells. Each macrocell is a configurable flipflop fed by an AND/OR matrix. Most macrocells are connected to an output pin allowing combinatorial or registered output. The larger EPLDs have embedded macrocells that have no output (for internal latches or shift registers) and expander terms (for more complex expressions). The combinatorial equations, flipflop configuration and whether a pin is input, output or both is programmed into the EPLD. Modifications can be made by erasing (using an ultraviolet light) followed by reprogramming. The EPLD used in this design is the Altera EPM5128 which consists of 128 macrocells in a 68 pin PLCC package.

The following paragraphs describe the circuit within the EPLD and refer to the EPLD schematic in Fig. A2 of Appendix A.

The clock divider section (in the extreme upper left corner) provides the clocks required on chip: 40 MHz, 20 MHz and the internal hop clock. Below this section is the upper byte of the status register (the lower byte is the FSK/Channel information) which reflects the current configuration switch positions (R8-R11), the status bits for 1 GHz Offset (R12), Synchronization

Controller command ready (R13), and the hop clock (R15). R14 always reads 0. The hop clock is chosen from the internal clock divider or from the Synchronization Controller interface external clock depending on the state of the Int/Ext configuration switch. The eight wide multiplexer to the right chooses between the upper byte of the status register and the upper byte of the latched Synchronization Controller command word. The high byte of status or the entire word of the Synchronization Controller command is then passed through a tristate buffer to connect to the TMS320C25 data bus.

The Synchronization Controller command word previously mentioned is latched on the rising edge of the SStrobe line. The rising edge also causes a flip-flop to be set indicating that a command is ready to be read by the C25. This flip-flop is automatically cleared when the command is read. The value of the flip-flop is reflected in the bit 13 of the status register as well as on the BIO pin of the C25. The latter is to exploit the fastest possible bit test on the C25 for rapid response during Mode Sync.

The lower left corner of the EPLD schematic covers the decoding of the various addresses. The port assignment can be found in Table 4. The unused ports are available for future expansion. Note that only the lower two bits of the port address are decoded.

A read from port 1 only puts out the upper 8 bits of the status. The lower 8 bits come from the FSK/Channel interface whose tristate buffer is enabled by this same read (Rd_FSK line). The purpose of the Clkout1 pulse is to provide an early rising edge that, when delayed by the propagation delay of the EPLD, will rise while the data is still valid during the write cycle. This edge is required by the synthesizer latches. For more details on this reasoning and how it affects modifications to the design see the following section on timing considerations. To further align the synthesizer latches, both strobes are delayed by 50-100 ns through a pair of 2-bit shift registers clocked at 20 MHz.

In the centre of the schematic is the logic necessary for the strobe which causes the synthesizer to change frequency (as opposed the on-board synthesizer latches that hold the frequency to the synthesizer). There are two possible strobe timings selected by the XF pin (external flag output, a general purpose output pin that is used here to select the strobe) of the C25: immediate and hop-edge. The immediate strobe is used during the Sync Mode and causes the synthesizer to change frequencies as soon as a value is written to the Synthesizer High Latch. The hop-edge strobe is used in normal Run Mode and causes the frequency to change at the rising edge of the hop.

The rising edge of the strobe (of the type selected) enables the shift register to clock out a pulse of the duration and shape necessary for the Comstron synthesizer. The enabling line is reset when the pulse has been shifted out.

3.5 Timing Consideration

A problem occurred when using the EPLD to strobe the onboard latches. The C25 holds the data for 30 ns (minimum) before and for 15 ns (minimum) after the write strobe. Unfortunately, the propagation delay of 30 ns in the EPLD means the strobe to the latch would be up to 15 ns late. Therefore, instead of using the NStrb pulse from the C25, the Clkout1 pulse (gated by the write line) was used because this combination precedes the strobe pulse by 25 ns. When this is delayed by 30 ns it produces a rising edge within the period for which the data lines are valid.

The onboard latch strobe portion of the design is heavily dependent on the speed and propagation times of the C25 and the EPLD. As the EPLD speed is increased, there will be less and less setup time for the latches. If the C25 was to speed up (faster clock speed or lower propagation times), the design would not work. If either the C25 or the EPLD were replaced by chips with longer propagation times, the synthesizer latch strobes would have to be redesigned because the Clkout1 pulse is no longer in the right place.

4. Conclusions

4.1 Summary

The author's task was to design and build a Hopping Synthesizer Controller to drive a Comstron synthesizer for a frequency hopped communications system. This was accomplished and the board was successfully tested.

Flexibility was a key aspect of the board design. This one board can be used in any of the four locations within a hopping system that use a synthesizer. The use of a digital signal processor and programmable read-only memories permit simple modification of software algorithms such as random number generation, frequency calculations or synthesizer frequency word formatting. The use of an EPLD allows simple and rapid modification of the hardware interface.

The Hopping Synthesizer Controller was implemented with the new generation EPLDs that featured more complex and denser building blocks. These devices were found to be very useful in replacing the numerous small integrated circuits normally required for any design. The design and programming of the EPLD were done on a personal computer simplifying design changes. Substituting one EPLD for all the usual support integrated circuits meant rapid development of the hardware as well as easy customization of the hardware interface which simplified the software requirement.

The finished board was tested and, as predicted, the processor was idle for about one-half of the time. At maximum speed, the board will support hop rates as high as 25000 hops/s.

4.2 Future Work

It is important that section 3.5, which dealt with timing considerations, be read prior to any attempt to substitute faster components or otherwise modify the hardware.

The board was designed to be flexible and capable of expansion. It is now fully functional yet there is plenty of capacity remaining on the board. 85% of the ROM and RAM are unused while the EPLD has 20% of it's capability available. There is also board space for additional interface logic and another set of ROM or RAM chips. The processor itself is working at slightly over half of it's capability.

The board could accommodate, without hardware modification, different synthesizers. The board currently supports only the Comstron synthesizer. Some steps have been taken in the design for other synthesizers. One of the DIP switches is a synthesizer configuration switch to indicate a synthesizer type other than Comstron. If it is set at Comstron, the software executes the Comstron frequency word formatting routine. When the switch is in the other position, all

calculations are done only up to the word formatting. At this point there is no formatting routine for the other synthesizer, so no further calculation are made. To allow the board to work with the Scitek synthesizer (the synthesizer being used by CRC), a custom cable would have to be built and the appropriate software formatting routine would have to be added. This is because the Scitek has a different connector and requires a special binary format for the frequency specification whereas the Comstron uses a BCD format. The routine CALC currently loads a zero frequency when the Scitek is selected. Instead CALC should invoke the Scitek conversion routine (yet unwritten) that would convert the binary frequency in the accumulator to the 32-bit format required by the Scitek. This routine would be similar in function to BCD32, the binary to BCD conversion routine for the Comstron.

Currently, the random number generation is done by a multiplicative linear congruential pseudo-random generator that has been shortened to repeat after approximately four seconds. While this is good for development and testing of link hardware, there are two simplifications that would have to be changed to do a full test. The first change is to allow the random sequence to achieve it's full length. This would be accomplished by removal of the code that forces the sequence to reinitialize after four seconds. The other change to consider is the type of random number generator. There is no known way of jumping ahead in time for the multiplicative linear congruential generator. In the current software, each intermediate random number is generated - a time consuming process for long sequences. Another type of random number generator would be required such as a PN sequence generator. This would require the rewriting of the three RAN routines (RAN - generates random number, RANINI - initialization, and RANJMP - move forward in the random sequence).

References

[1] D.G. Carta, "Two Fast Implementations of the 'Minimal Standard' Random Number Generator, *Communications of the ACM*, Vol. 33, No. 1, pp. 87-88, January 1990.

[2] *Operation and Maintenance Manual; Series FS2000 Frequency Synthesizer*, Comstron Corp., 1987.

[3] *Operation and Maintenance Manual; Series FS2000A/B Frequency Synthesizer*, Comstron Corp., 1987.

[4] *Second-Generation TMS320 User's Guide*, Texas Instruments Inc., 1987.

List of Abbreviations

BCD	Binary Coded Decimal
BW	BandWidth
C25	TMS320C25 Digital Signal Processor
CRC	Communications Research Centre
DIP	Dual Inline Package
DND	Department of National Defence
DREO	Defence Research Establishment Ottawa
EHF	Extremely High Frequency
EPLD	Erasable Programmable Logic Device
EPROM	Erasable Programmable Read Only Memory
FSK	Frequency Shift Keying
h	Suffix indicating a hexidecimal number
I/O	Input/Output
LO	Local Oscillator
PLCC	Plastic Leadless Chip Carrier
RAM	Random Access Memory
ROM	Read Only Memory
RXF	Reset eXternal Flag command
SATCOM	Satellite Communications
SXF	Set eXternal Flag command
sync	Synchronization
synth	Synthesizer
TTL	Transistor Transistor Logic
XF	eXternal Flag

Appendix A: Schematics

1. General

This appendix contains the schematics of the Hopping Synthesizer Controller board as well as the internal schematic of the EPLD and user macros. Details on location and description of the components can be found in the parts list and board layout of Appendix B.

2. EPLD Schematics

The EPLD is an Erasable Programmable Logic Device that emulates a collection of TTL logic. This is programmed using the MAX+Plus software from Altera. The EPLD schematic, Fig. A2, includes several TTL macros taken from the MAX+Plus TTL Macro Library. Some of the macros in the EPLD schematic are custom macros whose schematics are given in Fig. A3-A6. These macros are:

DIV2	Frequency divide by 2
DIV25	Frequency divide by 5 and 25
SHIFT2	2-bit D-FF shift register
81MUXB	8 wide 2:1 multiplexer, bus form

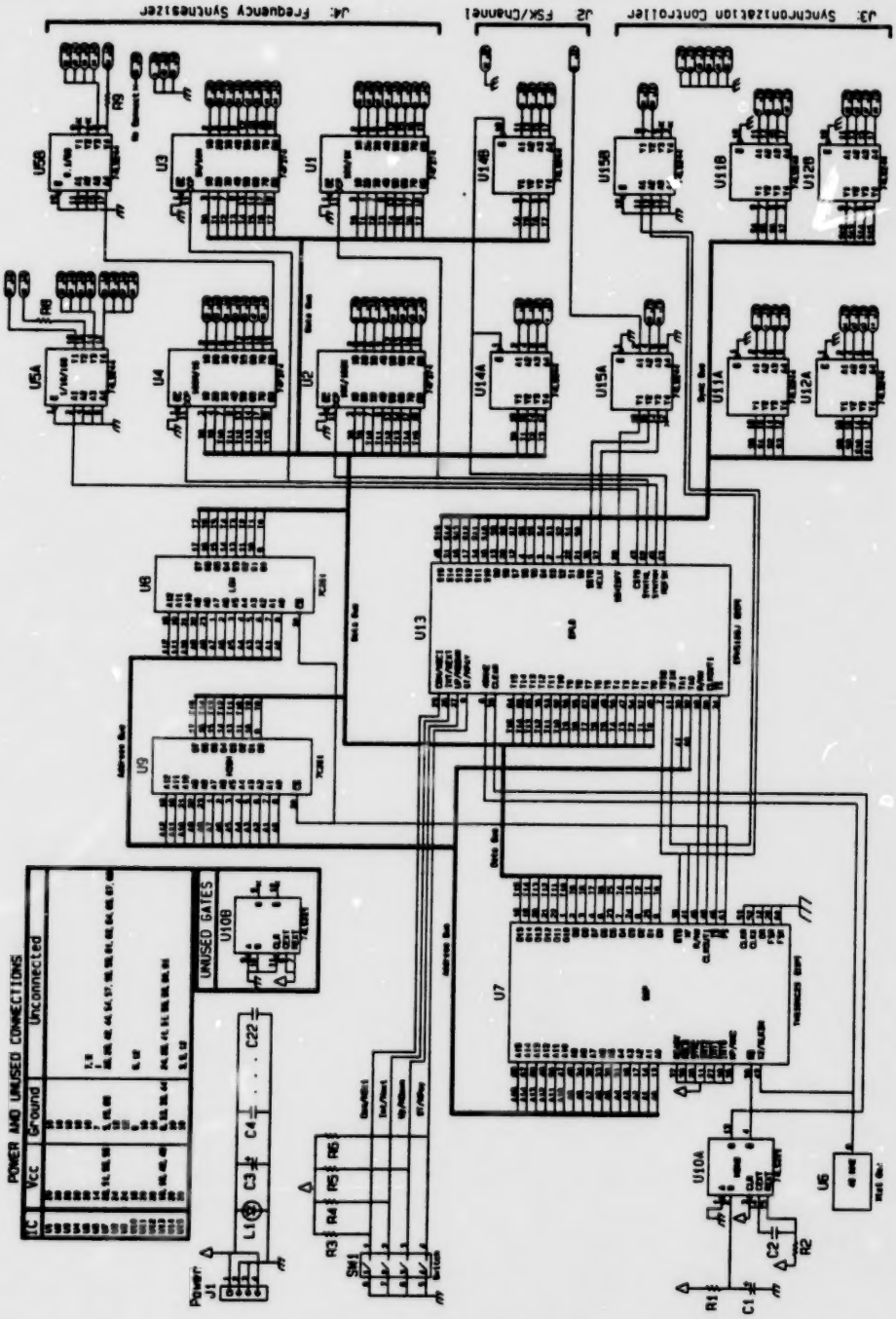


Fig. A1. Hopping Synthesizer Controller board schematic.

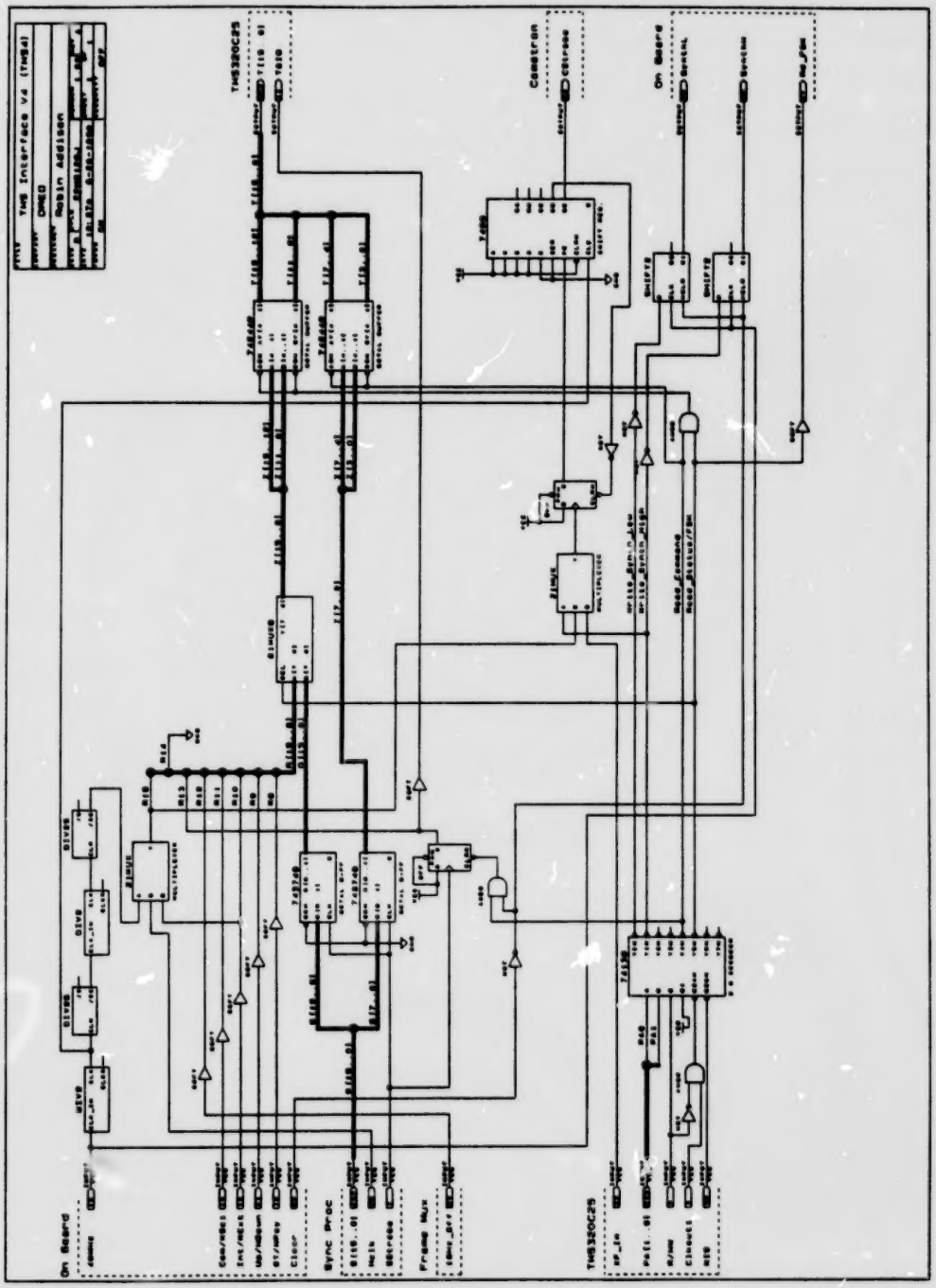


Fig. A2. TMS interface EPLD schematic.

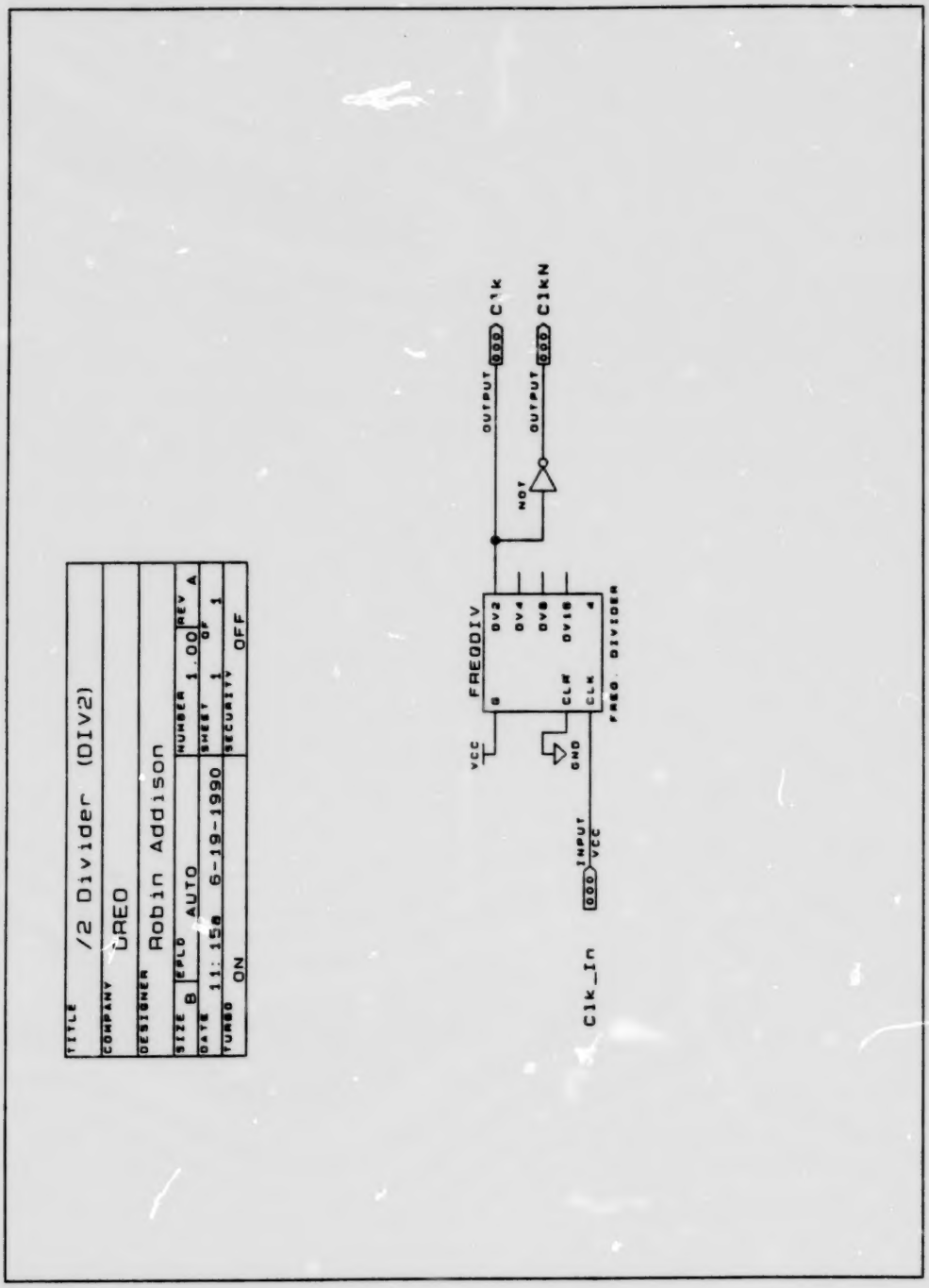


Fig. A3. DIV2 EPLD macro schematic.

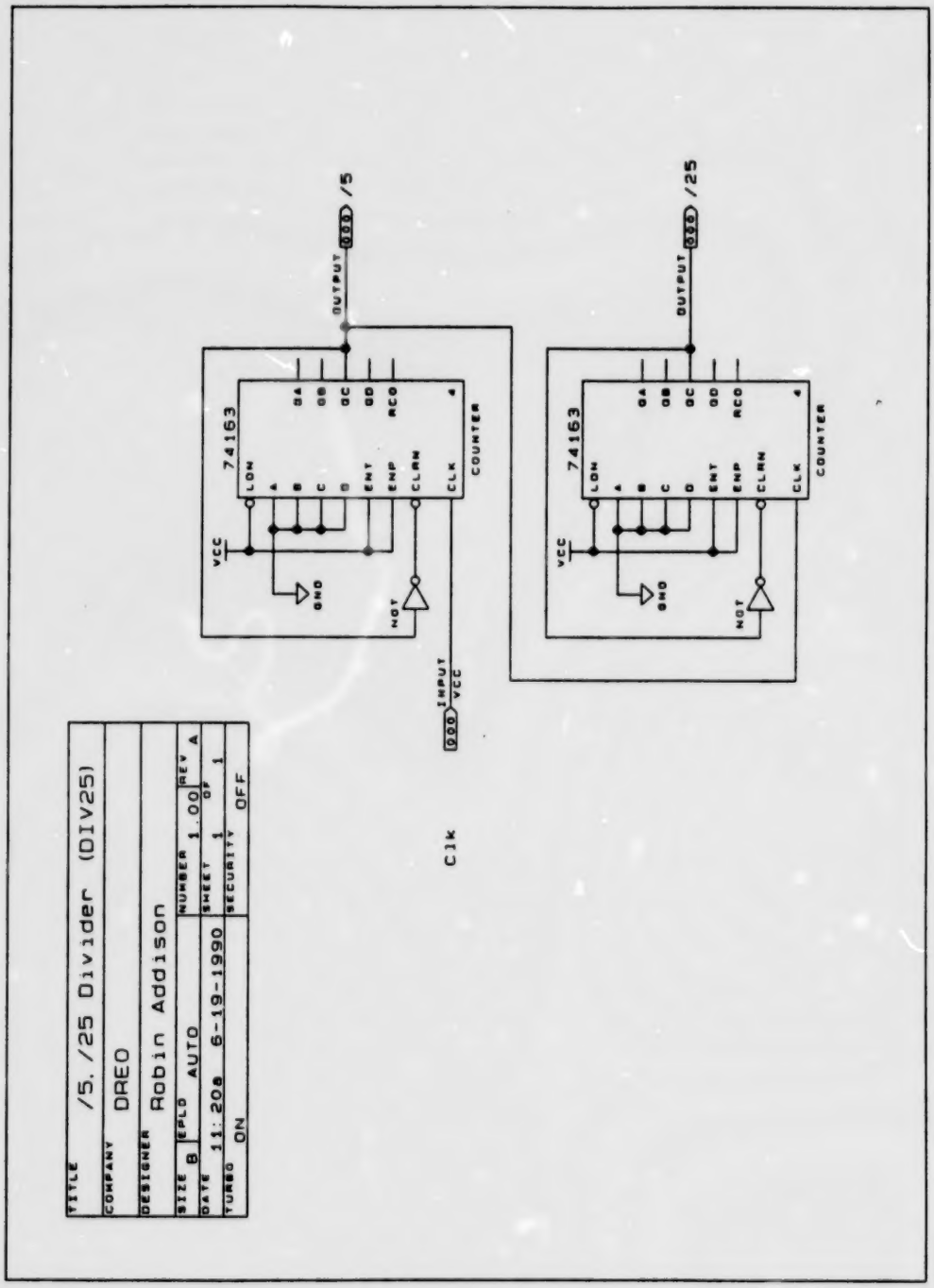


Fig. A4. DIV25 EPLD macro schematic.

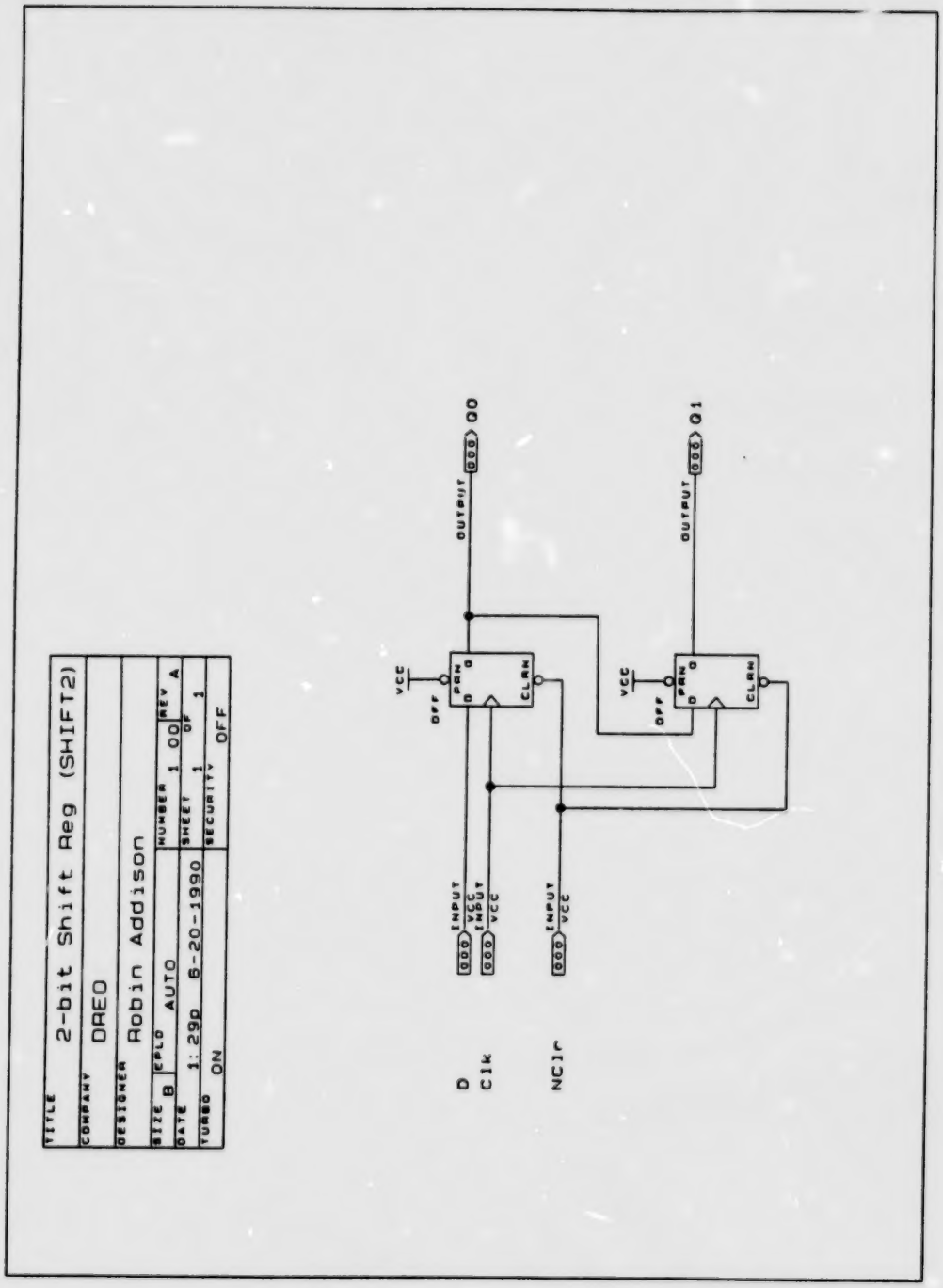


Fig. A5. SHIFT2 EPLD macro schematic.

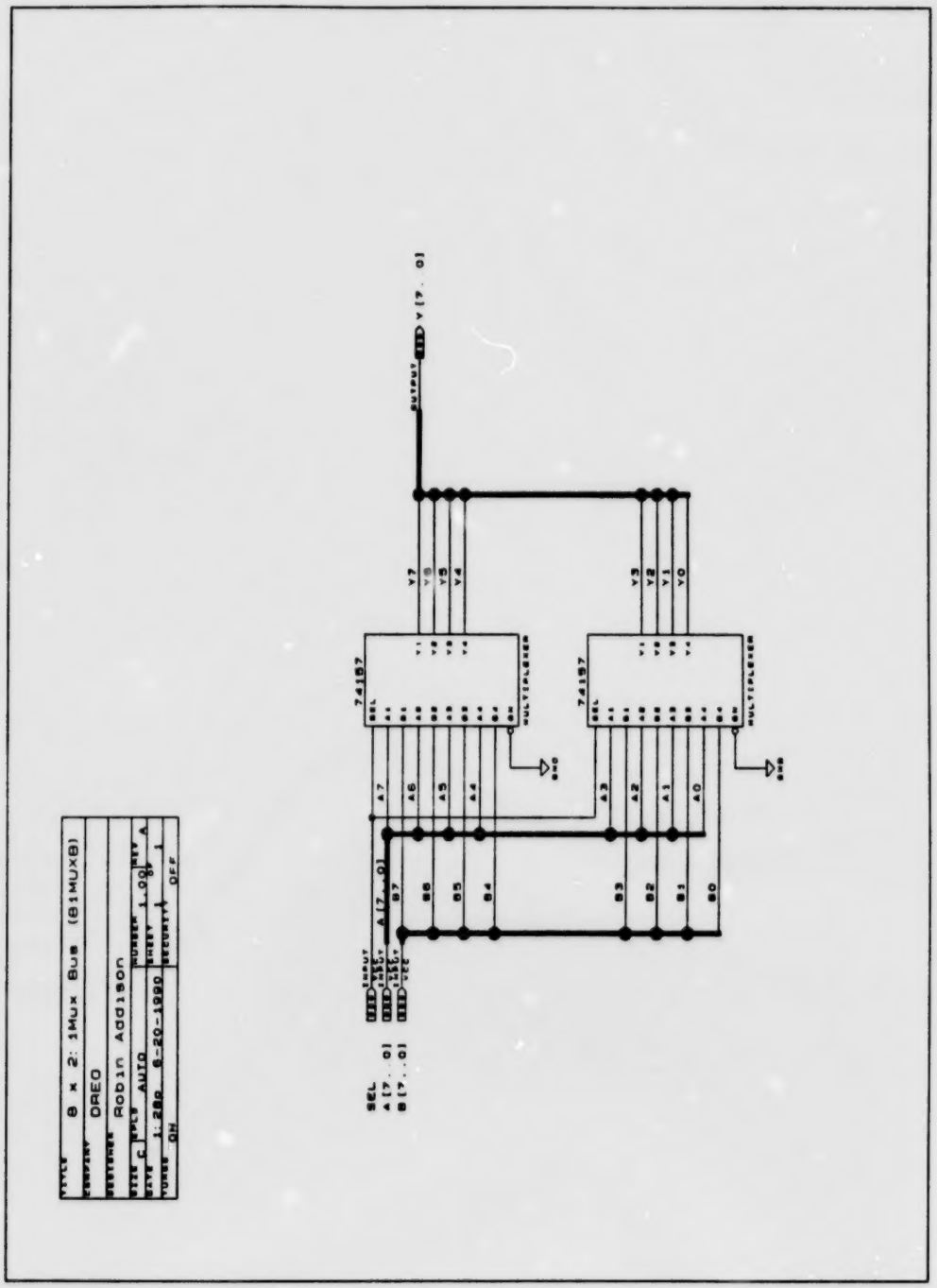


Fig. A6. 81MUXB EPLD macro schematic.

Appendix B: Parts List and Board Layout

1. Layout Description

The board layout of the Hopping Synthesizer Controller is shown in Fig. B1. Note that the board layout is roughly to scale. All parts are standard off-the-shelf with the exception of the EPLD (U13) and the program EPROMs (U8,U9) which must be programmed. In the layout, the driving factor was keeping the high speed lines as short as possible. This prompted all the components that connected directly with the TMS320C25 (U7) to be placed around it. There was space left beside the program EPROMs (U8,U9) to allow for either extra RAM or extra EPROMs. The upper left hand corner has room for additional TTL chips in the event of interfacing problems with different synthesizers. As well, there is unused capacity in the EPLD (U13) which could be used without modifying the board layout.

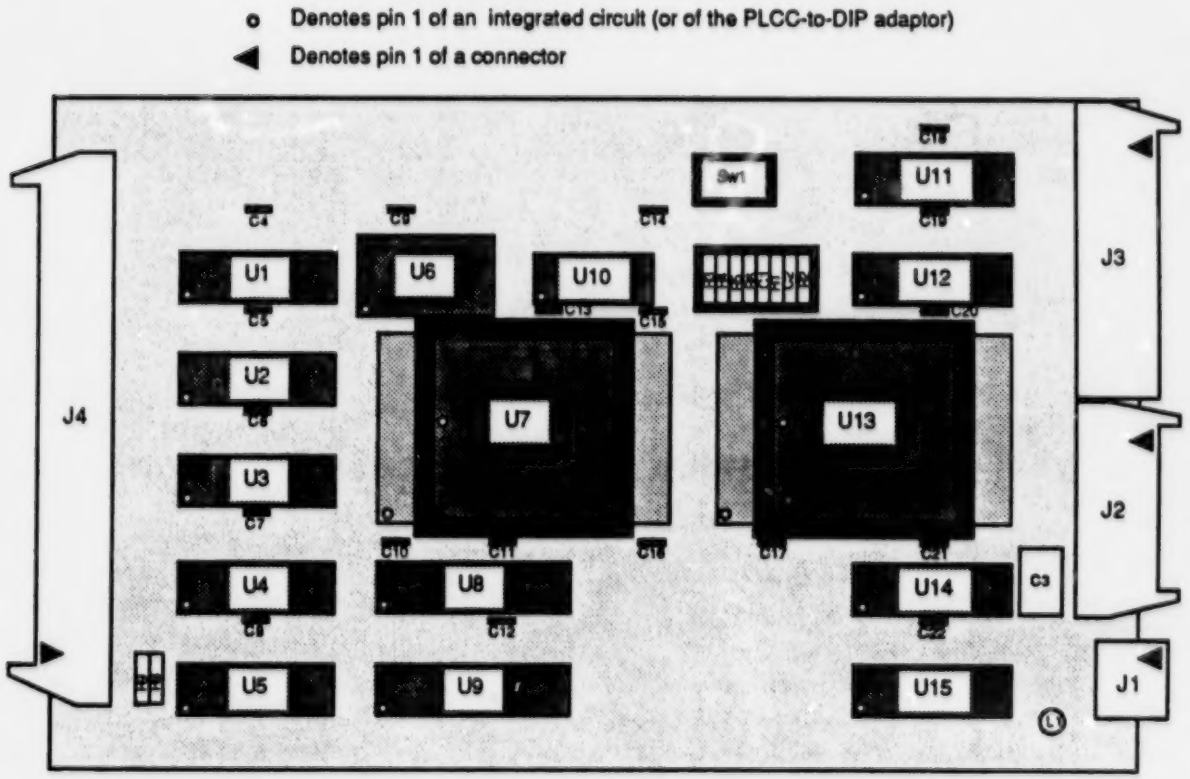


Fig. B1. Board layout.

2. Parts List

2.1 Integrated Circuits

U1-U4	74F374	Octal D-flipflops
U5	74LS244	Octal line drivers
U6	F1100E	Fox 40 MHz crystal oscillator
U7	TMS320C25FNL	TI Digital Signal Processor
U8,U9	7C261	Cypress 8k x 8 EPROM, 35 ns access
U10	74LS221	Dual monostable multivibrators
U11,U12	74LS244	Octal line drivers
U13	EPM5128JC-2	Altera EPLD, 128 macrocells, 30 ns prop
U14,15	74LS244	Octal line drivers

2.2 Discrete Components

R1	120 kΩ, ¼ W
R2	120 kΩ, ¼ W
R3-R7	10 kΩ, ¼ W
R8,R9	220 Ω, ¼ W
C1	15 µF, tantalum, 35 V
C2	0.0 1µF
C3	47 µF, tantalum, 20 V
C4-C22	0.1 µF
L1	LED, 5 V, 12 ma
Sw1	DIP switch, single throw, 4 wide

2.3 Connectors

J1	4 pin connector, right angle, single row, 0.1" spacing
J2	10 pin connector, right angle, double row, 0.1" spacing
J3	26 pin connector, right angle, double row, 0.1" spacing
J4	50 pin connector, right angle, double row, 0.1" spacing

2.4 Sockets

U7 and U13 reside in 68 pin PLCC to 0.1" x 0.3" adapters. The speedwire board does not support PLCC packages because the board has rows 0.3" apart with pins at 0.1" spacing. The adapter converts the PLCC pinout to the 0.1" x 0.3" pinout. Since this spacing is used mostly for DIPs, this adapter will be referred to as a PLCC to DIP adapter (despite the fact that there are more than 2 rows of pins). A diagram of the adapter is given in Fig. B2. The

mapping of PLCC pin numbers to DIP pin numbers is given in table B1. Note that the pin numbers used in the schematic are the DIP pin numbers.

Table B1. PLCC to DIP pin number mapping.

DIP	PLCC	DIP	PLCC	DIP	PLCC	DIP	PLCC
1	7	18	2	35	1	52	63
2	8	19	3	36	65	53	64
3	9	20	4	37	66	54	60
4	11	21	5	38	67	55	61
5	10	22	6	39	68	56	62
6	12	23	13	40	58	57	59
7	14	24	15	41	56	58	57
8	16	25	17	42	54	59	55
9	18	26	19	43	52	60	53
10	20	27	21	44	50	61	51
11	22	28	23	45	48	62	49
12	24	29	25	46	46	63	47
13	26	30	32	47	38	64	45
14	28	31	31	48	37	65	43
15	27	32	34	49	40	66	44
16	30	33	33	50	39	67	42
17	29	34	36	51	35	68	41

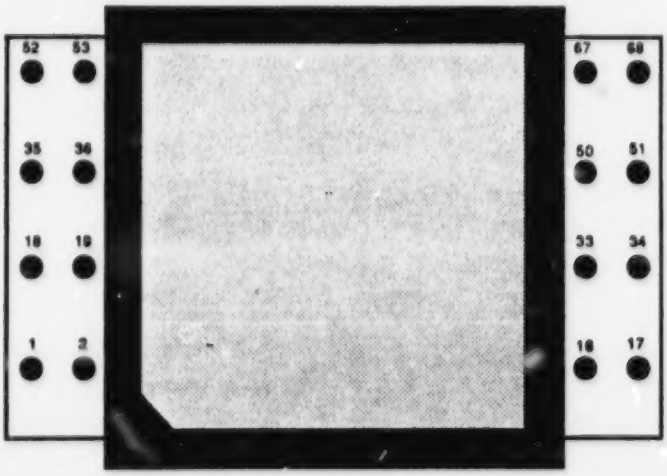


Fig. B2. Top view of PLCC to DIP adaptor.

Appendix C: Synchronization Controller Interface

1. General

The Hopping Synthesizer Controller must be under command of the Synchronization Controller for synchronization and for normal setting of timing, hopping, doppler and modulation parameters. During downlink synchronization, it is also necessary for the Hopping Synthesizer Controller to precompute five sync hop values and switch them rapidly under control of the Synchronization Controller. The following sections deal with the interface between the Synchronization Controller and the Hopping Synthesizer Controller.

2. Hopping Synthesizer Controller Commands

There are five types of commands for the Hopping Synthesizer Controller that consist of one or three 16-bit words. They are the Load, Set, Go, Change Mode and Debugging commands which are described below. The parallel port is only 16 bits wide, and is checked only once per hop in Run Mode and more often in other modes. Thus all commands will be processed at a rate of at least one 16-bit word per hop.

a. Load These commands load the values used in frequency computations. Each Load command consists of the command word followed by the lower 16-bit word and then the upper 16-bit word of a 32-bit longword. Load commands are not normally required unless values other than the defaults are needed. Note that one must always load the time before Sync Mode or Run Mode. See the Calculation Parameter Values section in this appendix for the default values, valid ranges and formats. Do not use this command in Sync Mode as it may cause indeterminate results. The Load commands include:

(1,2)	LOAD BCD	0300h	Loads BCD frequency
(1)	LOAD SEED	0302h	Loads Random Number seed
(1)	LOAD PN	0304h	Load PN sequence value
	LOAD PNSCALE	0306h	Load PN scale value (hop bandwidth)
(1)	LOAD FSKCHAN	0308h	Load FSK tone and Channel number
	LOAD FCSPACE	030Ah	Load FSK tone spacing
	LOAD BASE	030Ch	Load base frequency of hopping range
	LOAD OFFSET	030Eh	Load doppler guard offset from base
	LOAD DOPF	0310h	Load doppler factor
	LOAD LO	0312h	Load final LO frequency
(3)	LOAD TIMELO	0314h	Load low 32 bits of time
	LOAD TIMEHI	0316h	Load high 32 bits of time

Notes:

(1) These LOADs are only used for debugging and are included here only for

sake of completeness. Use other than for debugging will not result in any practical outcome.

(2) The BCD frequency is immediately sent to the frequency latches and will be latched by the synthesizer either immediately (in Sync Mode or if XF is set) or at the next hop (in Run or Stop Modes with XF reset).

(3) Loading the TIMELO automatically clears the TIMEHI. This saves a second load when the upper 32 bits are zero. To load a 64 bit time, first load low and then load high.

b. Set This command causes the default values (see Values section) to be reloaded from the ROM. This would be done on powerup or after the values are corrupted. Do not use this command in Sync Mode as it may cause indeterminate results. The only Set command is:

SET DEFAULTS	0400h	Reset default values

c. Go These commands are used to switch rapidly from one precomputed frequency to another. When the processor is put in Sync Mode, it precomputes the outputs for five different frequencies corresponding to sync hops. They are numbered 0 to 4 in chronological order, but frequency 4 is in a different sync hop burst from frequencies 0 to 3. Note the Change Sync Mode command implies a GO 0 once the computations are complete, so the only time that GO 0 is used is when restarting after a false alarm (occurs when the first sync hop is detected but the later sync hops are not found). These commands will be acted upon in less than 2 µs. The Go commands are only used in Sync Mode and will be ignored in all other modes. The Go commands are:

GO 0	0060h	Go to 1st sync hop frequency
GO 1	0062h	Go to 2nd sync hop frequency
GO 2	0064h	Go to 3rd sync hop frequency
GO 3	0066h	Go to 4th sync hop frequency
GO 4	0068h	Go to 5th sync hop frequency

d. Change Stop Mode This command is used to stop all hopping and to force the Hopping Synthesizer Controller into a known state. No matter what state the processor is in, three Change Stop Mode commands will force it into Stop Mode. The extra two are required in case a load command had been sent previously and the data portion was omitted. Change Stop Mode commands can be used in any mode although they are ignored in Stop Mode. The Change Stop Mode command is:

STOP MODE	0000h	Change Stop Mode

e. Change Sync Mode This command causes the processor to enter a tight loop waiting for

Go commands or a change to Run or Stop Mode. In Sync Mode, all Go commands cause an immediate change in the synthesizer's frequency. Because of the speed needed to respond to these commands, the processor's code has been simplified to eliminate error checking. This means that illegal commands will cause the processor to enter an indeterminate state. **In Sync Mode, Change Sync Mode commands or load/set commands will give indeterminate results and possibly require the processor to be hardware reset.** Use only Change Run Mode, Change Stop Mode or Go commands in Sync Mode. The Change Sync Mode command causes the processor to precompute the five sync hop frequency values, sets the synthesizer to the first value (equivalent to GO 0) and then enters the tight loop. The Change Sync Mode command is:

SYNC MODE	4000h	Change Sync Mode

f. Change Run Mode This command causes the processor to enter Run Mode. When going from Sync Mode to Run Mode, the fifth sync hop value is loaded into the synthesizer and the computation for the next frequency value is initiated. This value is then output to the synthesizer, but not latched until the next hop. The compute-output-latch cycle continues indefinitely once per hop from then on. In Run Mode, all commands are allowed, though change to Run Mode commands and Go Commands are ignored. The Change Run Mode command is:

RUN MODE	8000h	Change Run Mode

g. Debugging Commands The debugging commands are used during the debugging stage of the hardware and software. They can also provide functionality not typically needed during normal operation. The debugging commands include:

RXF	0500	Reset XF, latch synth on hop clock
SXF	0501	Set XF, latch synth on new frequency
LOAD BCD	0300	Send BCD to latches (see LOAD commands)
READ <addr>	2mmm	Read address mmm (see below)

Resetting and setting the XF (external flag) changes the time at which the synthesizer latches the new frequency. During Sync Mode (XF set), the synthesizer latches a new frequency word as soon as it is output by the controller. In Run Mode, where XF is reset, the synthesizer latches a new frequency word at the rising edge of the hop clock. The RXF and SXF commands allow the user to alter the default in Run Mode. Note that RXF and SXF are illegal commands when the Hopping Synthesizer Controller is in Sync Mode.

The READ command is a special command that causes the Hopping Synthesizer Controller to output an internal RAM value serially through the Sync line. The READ command includes the address to be read as the low 12 bits of the command. Once the command has been sent, the Hopping Synthesizer Controller prepares to output the bits (high bit to low bit). The bits are clocked by any commands except the Change Stop Mode command (the simplest way is to repeat the READ command) sent to the Hopping

Synthesizer Controller. There are only 16 useful bits to be clocked but no error is generated if more or less are clocked. Once the bits are clocked out, a Change Stop Mode command (no other commands work) should be sent to return the Hopping Synthesizer Controller to Stop Mode.

3. Calculation Parameter Values

The meaning of the calculation parameters used are described below along with their default values and ranges for the Ground Terminal Uplink configuration. All calculations are done fixed point with the units being 100 Hz. The default values are given in Table C1.

a. BCD (in 100 Hz)
This is the value sent to the synthesizer. It is normally calculated each hop but can be loaded in Stop Mode to achieve a CW signal of the desired frequency.

b. SEED
This is the value of the random number seed. It is reinitizialized when entering Sync or Run Mode, so changing this value in Stop Mode has no effect on the hopping pattern. Changing this value in Run or Sync Mode is not recommended.

c. PN
This is the value of the PN sequence and would be used to override the internal sequence generator. Maximum value is 16777215. This location is currently not in use but is reserved for supporting an external sequence generator.

d. PN Scale ($x2^{23}$)
This scale corresponds to the hopping bandwidth with guardbands. This is multiplied by PN to get a hopping frequency. Maximum value is 499500000 corresponding to a bandwidth of 99.9 GHz. For the GT Uplink the scale factor is 1.19090080261 giving a 2 GHz bandwidth with a 2 MHz guardband.

e. FSK/Channel
The lower 3 bits provide the tone number (0-7) while the upper 5 bits provide the channel number (0-31). Maximum composite value is 255 for tone 3 and channel 7.

f. FSK Spacing
This value is the FSK tone spacing and defaults to 40 kHz. The maximum value is 3.2768 MHz.

g. Base Frequency
This is the bottom of the frequency range used for all communications at this band. The maximum value is 99.9 GHz.

h. Guard Offset

This provides the guard area on the low edge of the bandwidth so doppler effects do not cause one to be outside the band. The maximum value is 99.9 GHz.

j. Doppler Factor ($x2^{32}$)

This is the multiplier that when multiplied by the frequency provides the doppler offset. This offset will then be added to the frequency. The maximum value is 42950 and the minimum value is -42950 with the smallest step of 1 corresponding to 9 Hz on 40 GHz. A negative value causes the doppler offset to be subtracted from the calculated frequency whereas a positive value cause the offset to be added.

k. Upconverter LO

This is the upconversion frequency mixed with the synthesizer output and as such it is subtracted from the final frequency before it is sent to the synthesizer. The maximum value is 99.9 GHz.

m. Time (low and high)

This is the number of hops since Time 0. It will be used as the time of the 1st sync hop when going to Sync Mode. This should normally be set by the Synchronization Controller rather than relying on the default value.

Table C1. Default values for calculation parameters.

Variable	GT Up	Pay Up	GT Down	Pay Down	Remarks
BCD	0	0	0	0	(debug only)
SEED	1	1	1	1	(debug only)
PN	0	0	0	0	(debug only)
PNSCALE	9990000	9990000	4977500	4977500	2/1 GHz BW
FSKCHAN	0	0	0	0	(debug only)
FCSPACE	400	0	0	0	40 kHz space
BASE	435000000	435000000	202000000	202000000	43.5/20.2 GHz
OFFSET	5000	5000	2500	2500	500/250 kHz
DOPF	0	0	0	0	
LO	430000000	430000000	192000000	192000000	43.0/19.2 GHz
TIMELO	0	0	0	0	
TIMEHI	0	0	0	0	

4. Hardware Interface

The interface consists of a 26-pin connector providing a 16-bit unidirectional parallel port with handshaking and a hop clock. The Sync line, normally used for handshaking, can be used to return serial data from the Hopping Synthesizer Controller.

D0-D15 — The sixteen data lines originate from the Synchronization Controller and are active high TTL lines.

Gnd — Digital ground.

HClk — The HClk (hop clock) line originates from the Synchronization Controller and is an active high TTL line. This clock signal should change from 0 to 1 at the beginning of the hop period and have a pulse high time of at least 30 ns. In Run Mode, this will be used to latch the frequency synthesizer.

Ready — The Ready line originates from the Hopping Synthesizer Controller and is an active high TTL line. The Ready line is normally 1; when the new data is latched the Ready line is forced to 0 until the Hopping Synthesizer Controller has read the value from the latch. The time for this to occur is less than one hop, is typically less than 40 μs and will be less than 2 μs in Sync Mode. The old data will be lost if the Hopping Synthesizer Controller is not ready during the strobe of new data.

Strobe — The Strobe line originates from the Synchronization Controller and is an active high TTL line. The Strobe line is normally 0 and when new data is stable, goes from 0 to 1 for at least 50 ns and then returns to 0. The data lines must be stable before and after the transition and will be latched on the rising edge of Strobe.

Sync — The Sync line originates from the Hopping Synthesizer Controller and is an active high TTL line. This line is normally 0 and goes to 1 when the processor, having completed all calculations, is in the tight loop and ready to respond immediately to Go commands. The Synchronization Controller should check this line to determine when the Hopping Synthesizer Controller is ready to change frequencies rapidly. This line returns to 0 when Change Run Mode or Change Stop Mode is commanded.

The Sync line is also used during a READ operation to provide the serial data out MSB to LSB. In this case it is clocked by sending a command (any command, other than Change Stop Mode, which is ignored). The new bit is valid when the Ready line goes back up after a command was sent. The Change Mode Stop command (all zeros) indicates

the end of the read and the Sync line then is use to indicate Sync Mode. The Sync line is a buffered version of the TMS320C25's XF output.

The pinout of the connector is given in Fig. C1 and the timing diagrams are given in Fig. C2.

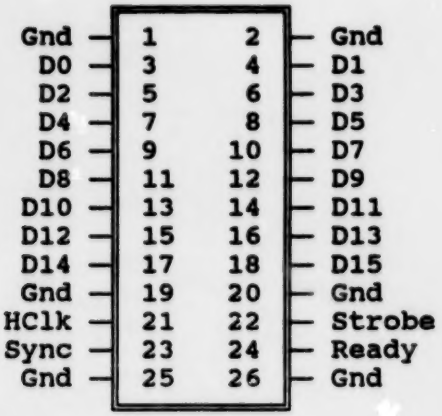
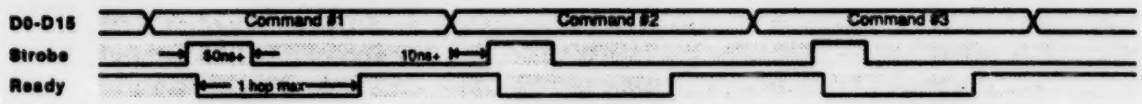
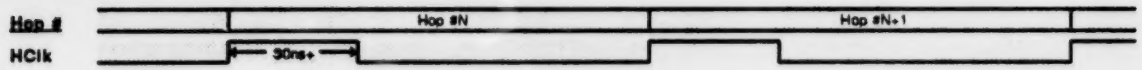


Fig. C1. Synchronization Controller interface pinout.

Command strobe timing

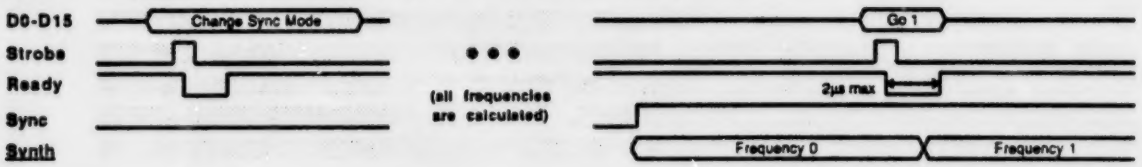


Hop Clock timing (Note that Hop # is the hop number and not a control line)



Changing to Sync Mode timing

(Note that Synth is the synthesizer output and not a control line)



Changing from Sync to Stop Mode timing

(Note that Synth is the synthesizer output and not a control line)

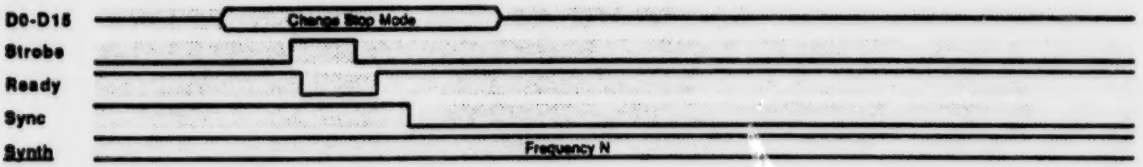


Fig. C2. Synchronization Controller interface timing diagrams.

Appendix D: FSK/Channel Interface

1. General

The FSK/Channel Interface connects the Frame Multiplexer to the Hopping Synthesizer Controller. The FSK modulation and channel selection for each hop are sampled by the Hopping Synthesizer Controller. An additional feature is a 1 GHz offset line that causes the hopper to do a hop within the band but different from hopping pattern. This will be used to simulate turning off the transmitter.

2. Hardware Interface

The interface consists of a 10-pin connector providing 3 bits for FSK tone specification, 5 bits for channel number and 1 line for 1 GHz offset. There is no handshaking; all 9 lines are sampled at the rising edge of the hop clock for the next hop. On the rising edge of the hop clock of hop #1, the 9 lines for hop #2 are sampled. The hop clock is derived internally (if in internal clock mode - see configuration switches in Appendix F) or externally through the Synchronization Controller Interface (see Appendix C). The pinout can be found in Fig. D1 and the pin descriptions follow.

FSK0-FSK2 — The three active high TTL lines originate from the Frame Multiplexer and are sampled on the rising edge of the hop clock, 1 hop ahead. They indicate the 8-ary FSK symbol with FSK0 as the LSB and FSK2 as the MSB. The symbols range from 0 (000 binary) to 7 (111 binary) where 0 is the lowest frequency tone.

Chan0-Chan4 — These five lines originate from the Frame Multiplexer and are sampled on the previous hop's rising edge of the hop clock. They indicate the channel number and are active high TTL. Chan0 is the LSB and Chan4 is the MSB. The channel numbers range from 0 (00000 binary) to 31 (11111 binary) where 0 is the lowest frequency channel.

1GHzOff — This active high TTL line comes from the Frame Multiplexer and, when set, causes the 1 GHz bit of the synthesizer to be toggled. The effect is a hop within the hopping bandwidth, but not according to the hopping pattern, and can be used to simulate jamming or turning the transmitter off. This special effect is not normally used. This pin should be tied to ground when not in use. 1GHzOff is an active high TTL line which is sampled on the previous rising edge of the hop clock.

Gnd — Digital ground.

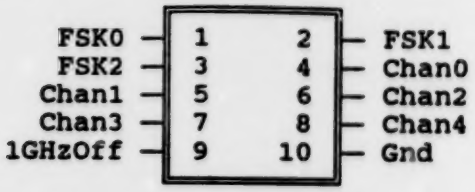


Fig. D1. FSK/Channel interface connector pinout.

Appendix E: Frequency Synthesizer Interface

1. General

The Hopping Synthesizer Controller drives the Frequency Synthesizer through a parallel port for full speed hopping. The interface is set up for direct connection to a Comstron synthesizer and could be connected to other types if custom cables are made. The output is in BCD form with only eight decades controlled, the rest are forced to zero.

2. Comstron Synthesizer

The Comstron Synthesizer and it's interfacing requirements are documented in [2] and [3]. The synthesizer accepts a BCD frequency selection up to 18 GHz (on some models) with a resolution of 0.1 Hz. The models used in testing the board could cover from 0 to 4 GHz with rapid switching in less than 1 μs.

3. Comstron Cable

The cable connecting the Hopping Synthesizer Controller to the Comstron Synthesizer is a 50 pin ribbon cable with a header crimped on the controller end and a 50 pin D connector connected on the synthesizer end. Note the cable connecting the Comstron to the Hopping Synthesizer Controller should be no longer that 3 feet. This is detailed in [2] and [3].

4. Hardware Interface

The interface consists of a 50 pin connector made up of numerous frequency selection lines with one strobe line and some unused power lines. The pinout of the connector is shown in Fig. E1. The pinout by function is listed in Tables E1 and E2. The crimp D-connector at the Comstron follows a different pin numbering convention and the mapping of this connector to the board connector is given in Table E3. The description of the interface pins are given below:

frequency select

These 45 data lines originate from the Hopping Synthesizer Controller and are active high TTL lines. The Comstron loads these lines with four standard TTL loads. They are latched by the synthesizer on the rising edge of the Strobe line. They should be stable before this edge. Note that the 0.1 Hz, 1 Hz, 10 Hz and 10 GHz decades are tied to zero on the board and cannot be commanded.

The most significant bit of the 1 GHz decade (pin 24) and the least significant bit of the 10 GHz decade (pin 10) are both driven through 220 Ω resistors because of incompatibilities between two models of Comstron. One model places +5V on pin 24 while

it is an input on the other. Pin 10 has TBD (assume the worst) for one model and input for the other. These resistors enable the interface to work with either model. Larger resisters coupled with the capacitance of the interface cable cause excessive time delays.

Strobe — This active high TTL line originates in the Hopping Synthesizer Controller; the rising edge causes the synthesizer to latch the frequency selection lines. It must rise after the frequency selection lines are stable and stay high for a minimum of 150 ns. The trailing edge must occur 20 ns before the frequency selection changes. In this implementation, the rising edge occurs 50 ns after the frequency selection is valid and stays high for 200 ns.

+5V — This is a five volt source from the Comstron. This line is unconnected on the Hopping Synthesizer Controller board.

Logic Ground — Tied to digital ground on the board.

Chassis Ground — Tied to digital ground on the board.

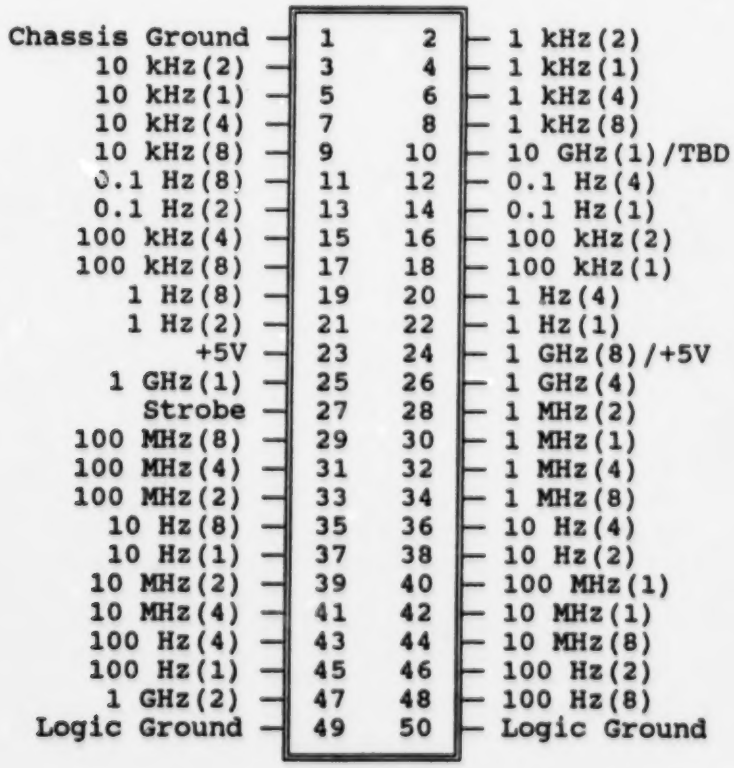


Fig. E1. Frequency synthesizer interface connector pinout.

Table E1. Frequency selection interface connector pin numbers.

Decade	Bit value			
	8	4	2	1
10 GHz*	-	-	-	10
1 GHz	24	26	47	25
100 MHz	29	31	33	40
10 MHz	44	41	39	42
1 MHz	34	32	28	30
100 kHz	17	15	16	18
10 kHz	9	7	3	5
1 kHz	8	6	2	4
100 Hz	48	43	46	45
10 Hz*	35	36	38	37
1 Hz*	19	20	21	22
0.1 Hz*	11	12	13	14

* these decades are tied to zero

Table E2. Non-frequency selection interface connector pin numbers.

Function	Pin Number(s)
Strobe	27
+5V	23
Logic Ground	49, 50
Chassis Ground	1

Table E3. Mapping of board connector to Comstron connector

Hop Synth Cont 50 pin Header	Comstron 50 pin D	Hop Synth Cont 50 pin Header	Comstron 50 pin D
1	1	26	38
2	26	27	14
3	2	28	39
4	27	29	15
5	3	30	40
6	28	31	16
7	4	32	41
8	29	33	17
9	5	34	42
10	30	35	18
11	6	36	43
12	31	37	19
13	7	38	44
14	32	39	20
15	8	40	45
16	33	41	21
17	9	42	46
18	34	43	22
19	10	44	47
20	35	45	23
21	11	46	48
22	36	47	24
23	12	48	49
24	37	49	25
25	13	50	50

The difference in pin numbers between the two connectors in this table is due to the non-standard pin numbers of the D-connector when it is crimped to a ribbon cable.

Appendix F: User's Guide

1. General

This appendix covers the information required to install, configure and use the Hopping Synthesizer Controller board. The latter portion consists of a description of the modes of operation and some examples of commanding the board through the Synchronization Controller Interface. The examples given are: initialization of the board, reading onboard RAM values, synchronization, and setting a CW frequency. Further details of this interface, including the opcode values, can be found in Appendix C.

2. Installation

This board drives a COMSTRON frequency synthesizer and according to the manual the ribbon cable connecting the two should be no more than three feet. Connect this cable to J4, the 50-pin connector on the left side of the board (interface described in Appendix E). The 5V power supply should be connected to J1, the 4-pin connecter on the lower right of the board. The pinout details for J1 can be found on the upper left of the board schematic in Appendix A.

If the board is being operated in a ground terminal it is necessary to connect the Synchronization Controller Interface (this is optional for the payload configuration). This 26-pin connector, J3, is found on the upper right side of the board.

The FSK/Channel connector, J2, is found in the centre right of the board and is used to connect the Frame Multiplexer. This connection is only made for the ground terminal uplink configuration. In all other cases, pins 9 and 10 of J2 should be jumpered. This forces the 1GHzOff line to the 0 or no offset state. If it is desired to simulate turning off the transmitter, pin 9 should be raised high before the rising edge of the hop for the next hop. See Appendix D for more details.

Once all connectors are made, the DIP switches must be configured.

3. Configuration

At the top centre of the board is a quad-wide DIP switch (SW1). This switch is used to configure the board for operation in any position in the simulator. The details are given in Table F1.

Table F1. Configuration DIP switch settings.

Switch	Selects	Position	TTL Value	Value
SW1-1	Synthesizer Type	Closed Open	0 1	Scitek (not implemented) Comstron
SW1-2	Hop Clock Source	Closed Open	0 1	External (J1 pin 23) Internal
SW1-3	Link Type	Closed Open	0 1	Downlink Uplink
SW1-4	Board Location	Closed Open	0 1	Payload Ground Terminal

The four switches are described in the following paragraphs.

Synthesizer Type: the board and software have the potential to control two types of synthesizer. The Comstron synthesizer is the one being used in the simulator and is thus supported. Plans were made to support the Scitek synthesizer as the alternate type, but the implementation is not yet complete in software. This switch should always be in the Comstron position.

Hop Clock Source: in the simulator, the hop clock will be supplied externally from the Sync Controller interface. For debugging or for stand-alone operation, position the switch to the internal hop clock.

Link Type: this switch is used with the Board Location switch to determine which end of which link is being serviced. In the downlink position, it selects the narrower hopping bandwidth with no FSK modulation. The uplink position selects a wider hopping bandwidth and also allows FSK modulation if the Location switch is in the Ground Terminal position.

Board Location: this switch is used with the Link Type switch to determine which end of which link is being serviced. If in the ground terminal position, this enables FSK modulation on the uplink and causes the board to powerup in the Stop Mode. The Payload position causes the board to powerup in the Run Mode.

Each of the four possibilities (uplink/downlink, payload/ground terminal) configurable by the last two switches has a set of default values including hopping bandwidth, FSK modulation, base, offset and LO frequencies.

4. Hopping Synthesizer Controller Modes of Operation

The processor has three distinct modes of operation. At power up, the processor is normally in Stop Mode. The regular sequence would be to load in any required values, then

command it into Sync Mode and finally, when synchronization is achieved, command it into Run Mode. For a standalone hopper, the processor would power up in Run Mode. The details of the three modes follow:

Stop Mode

In this mode, the processor is waiting for commands. There are no updates to the synthesizer and there are no frequency calculations. This is the normal powerup mode. When one wishes to stop hopping, the processor should be commanded to Stop Mode.

If Change Stop Mode is commanded while in Sync or Run Modes, the result will be a CW signal at the last frequency.

In this mode, only load/set, Change Mode and debug commands can be given. Go commands will be ignored. The command Change Stop Mode when already in Stop Mode is ignored. This feature is used during initialization by sending multiple Change Stop Mode commands. (For more details see the section 5 on initialization.)

The response time in this mode is less than 10 μs.

Sync Mode

This mode is used when acquiring synchronization. The processor is in a tight loop waiting for commands to switch to precomputed frequencies.

In this mode, only the Go commands and the Change Mode commands (Run and Stop) are allowed. Commands to change to Sync Mode and Load/Set commands when in Sync Mode will give indeterminate results possibly requiring a hardware reset.

The response time in this mode is less than 2 μs.

Run Mode

The processor should be put into Run Mode when synchronization has been achieved. In this mode, the random hop frequency is computed and output once per hop.

In this mode, all commands except Go commands are permissable. All Go commands and the Change Run Mode command when already in Run Mode will be ignored.

The response time in this mode is less than one hop.

5. Hopping Synthesizer Controller Initialization

Initializing consists of getting the processor to a known state (Stop Mode), reloading the default values, loading any modifications to these values and the loading the time. The necessary commands are listed below the initialization steps. (The associated op codes can be found in the Appendix C.)

Step 1

Stop Mode - To ensure that the processor was not waiting for data words, one must send the command three times if the state of the processor is unknown. It is possible that a Load command, without associated data, was sent prior to this point. The two extra commands will provide the

data for the Load command and the third Change Stop Mode will be executed.

STOP MODE
STOP MODE
STOP MODE

Step 2 Set Default Values - If it is suspected that the values have been corrupted or if they are unknown, they should be reloaded from processor's ROM.

SET DEFAULTS

Step 3 Modifying Default Values - If it is desired to have some of the values different from the defaults, such as hopping bandwidth, then the Load command should be used to modify them.

LOAD PNSCALE
<pnscale: 16 least significant bits>
<pnscale: 16 most significant bits>

Step 4 Loading Current Time - The time defaults to 0 and this should normally be changed to the desired time. (Note that the LOAD TIMELO command also zeros the high 32 bits, so if only 32 bits of time are required, the LOAD TIMEHI can be omitted.)

LOAD TIMELO
<time: 16 least significant bits>
<time: 16 lower middle significant bits>
LOAD TIMEHI
<time: 16 higher middle significant bits>
<time: 16 most significant bits>

6. Reading Values

To examine internal RAM values within the Hopping Synthesizer Controller, the READ command can be used. For example, to examine the PNSCALE value at address 0306H, the following sequence of commands should be used.

Step 1 Setup for Read - In Stop Mode, the READ command is used to load the RAM value into the accumulator in preparation to be shifted out.

READ 0306

Step 2 Shift Out MSB - When a non-zero command (for example repeat the READ command) is sent, the Ready line is lowered, then the next bit is put out on the Sync line and finally the Ready line is set.

READ 0306
- wait for Ready
- read MSB from Sync line

Step 3-17 Shift Out Rest - Repeat step 2 for the other 15 bits.

Step 18 End the Read - A zero command (STOP MODE) is used to end the read command and return the Hopping Synthesizer Controller to Stop Mode.

STOP MODE

7. Synchronization

This section outlines the command steps involved in achieving synchronization using the Hopping Synthesizer Controller. First the desired time-of-day to start synchronization should be sent when the Hopping Synthesizer Controller is in Stop Mode. With the current random number scheme there is no way to jump ahead in the sequence so instead, the sequence is computed rapidly. It requires 2 μs to compute each hop so the desired time to start synchronization must include the computation time up to the last sync hop which is 5120 hops after the first. For example, to start at hop 10000, one should leave at least 30.24 ms [(10000 + 5120) x 2μs] of computation time before starting synchronization.

Once the time is set, the mode should be changed to Sync Mode which causes the Hopping Synthesizer Controller to go through the sequence up to the desired time, set the appropriate frequency on the synthesizer and then set the Sync Line. When the Sync Line is set, the Go commands can be used to rapidly (under 2 μs) switch to different synchronization frequencies. The following sequence of steps is used to start synchronization at hop 100 assuming that the Hopping Synthesizer Controller is in Stop Mode with default values set.

Step 1 Set Desired Sync Time - To start at time 100 (0064H), set low order time to 100 automatically zeroing high order time.

LOAD TIMELO
64H
00H

Step 2 Go to Sync Mode - This command should be given at least 1.044 ms ([100+5120] hops x 2 μs) before the desired time.

MODE SYNC

Step 3 Wait for Sync Line - The computations take approximately 2 μs per hop but the Sync Line indicates exactly when the computations have finished and when the synthesizer is at the first frequency f_{100}.

Step 4 Change Frequencies as Desired - Use the Go commands to change frequencies as required. The normal sequence is to change as soon as the sync hop is detected (which takes place one hop after the sync hop occurred.) Each sync hop, within a burst, is separated by a hop and the sync hop bursts occur every 5120 hops.

GO 1	(f_{102})
GO 2	(f_{104})
GO 3	(f_{106})
GO 4	(f_{5220})

Step 5 Changing to Run Mode - Assuming all sync hops were detected, the Change Run Mode must be made two rising edges of the hop clock before the transition occurs. GO 4 should be commanded anywhere after hop 108 (for this example) and RUN MODE should be commanded during the 5219^{th} hop. On the rising edge of the hop clock for hop 5220, f_{5220} will be relatched by the synthesizer and computations for hop 5221 will be made. Then on the next rising edge of the hop clock, f_{5221} will be latched and computations for 5222 will be made, etc. From this point on normal hopping occurs.

RUN MODE

8. Setting a CW Frequency

For testing purposes, it is sometimes advantageous to drive the hopping synthesizer to a fixed frequency. The Hopping Synthesizer Controller can do this in one of two ways: switch to CW frequency on next hop, or switch to CW frequency immediately. Both ways require the Hopping Synthesizer Controller to be in Stop Mode.

Switch on next hop:

Step 1 Ensure latch on hop - Normally the frequency synthesizer latches the frequency on the rising edge of the hop clock (XF reset). This step is only required if XF was changed previously.

RXF

Step 2 Send BCD value - Load the actual BCD value (note there is no doppler or

other calculations - this loads the latches directly).

LOAD BCD
<BCD 16 least significant bits>
<BCD 16 most significant bits>

Switch immediately:

Step 1 Ensure immediate latch - The normal state is for the Frequency Synthesizer to latch the frequency on the rising edge of the hop clock (XF reset). The XF must be first set to latch immediately.

SXF

Step 2 Send BCD value - Load the actual BCD value (note there is no doppler or other calculations - this loads the latches directly).

LOAD BCD
<BCD 16 least significant bits>
<BCD 16 most significant bits>

Step 3 Restore to latch on hop - When normal operation is to be resumed, XF must be reset to allow latching on the rising edge of the hop clock.

RXF

Appendix G: Software Listings

1. General

The software was generated on a COMPAQ Deskpro 386 running under COMPAQ DOS 4.01. The Texas Instruments TMS320C1x/TMS320C2x Assembly Language Tools were used for assembly, linking and reformatting. The make file used to generate the EPROM files follows:

```
MAIN.OBJ:       MAIN.ASM
        DSPA MAIN.ASM -l
CALC.OBJ:       CALC.ASM
        DSPA CALC.ASM -l
RAN.OBJ:        RAN.ASM
        DSPA RAN.ASM -l
MAIN.OUT:       MAIN.OBJ CALC.OBJ RAN.OBJ
        DSPLNK MAIN.CMD
MAIN.TAG:       MAIN.OUT
        DSPROM MAIN.OUT -t
MAIN.HI:        MAIN.OUT
        DSPROM MAIN.OUT -i
```

The command file used by the linker (MAIN.CMD) is given below:

```
 MAIN.OBJ      /* Main program                          */
 CALC.OBJ      /* PN -> BCD Frequency calculation      */
 RAN.OBJ       /* Random number generation             */
 -e MAIN       /* Entry point                          */
 -o MAIN.OUT   /* Executable file                      */
 -m MAIN.MAP   /* Map file                             */
MEMORY
{
  PAGE 0: VECTORS:  origin =   0H,   length =  03FH
          PROG:     origin = 1000H,  length = 0EF00H
  PAGE 1: DATA:     origin =  300H,  length = 0FD00H
}
SECTIONS
{
  IRUPTS: { } > VECTORS PAGE 0
  .text:  { } > PROG    PAGE 0
  .data:  { } > PROG    PAGE 0
  .bss:   { }           PAGE 1
}
```

The next three sections are the software listings for the main program (MAIN), the calculation routines (CALC) and the random number generation routines (RAN).

2. Main Program (MAIN.ASM)

```
        .length   55
        .width    120
        .global   MAIN
        .global   RESET,INT0,INT1,INT2,TINT,RINT,XINT,USER,BADINT
        .global   CALC,SETDEF,MUL32,BCD32,BCDINI,RANINI,RANJMP,RAN
        .global   ANSWER,SEED,A,ZERO,ONE,C0FFFF
        .global   PN,SCLPN,FC,FCDEL,FBASE,F0,DOPF,FLO,TLO,THI
        .global   X0,X1,Y0,Y1,Z0,Z1,Z2,Z3,SIGN
        .global   CMD,DATA1,DATA2,WPNT,STAT,MODE
        .global   HTAB,HEND,LTAB,LEND


BGT:    .set      7
BUP:    .set      6
BINTCK: .set      5
BCOMST: .set      4
BIGHZ:  .set      3
BBIO:   .set      2
BSPARE: .set      1
BHOP:   .set      0

CMDIN:  .set      0
STATIN: .set      1
SYNTHL: .set      0
SYNTHH: .set      1

MSTOP:  .set      0
MSYNC:  .set      4000H
MRUN:   .set      8000H

MGO0:   .set      0060H
MGO1:   .set      0062H
MGO2:   .set      0064H
MGO3:   .set      0066H
MGO4:   .set      0068H

MSETDF: .set      0400H

MRXF:   .set      0500H
MSXF:   .set      0501H

MLDANS: .set      0300H
MLDSED: .set      0302H
MLDPN:  .set      0304H
MLDPNS: .set      0306H
MLDFC:  .set      0308H
MLDFSP: .set      030AH
MLDBAS: .set      030CH
MLDOFF: .set      030EH
MLDDOP: .set      0310H
MLDLO:  .set      0312H
MLDTLO: .set      0314H
MLDTHI: .set      0316H

MREAD:  .set      2000H
MRDEND: .set      2FFFH

CPRETY: .set      1111H

        .sect     "IRUPTS"
RESET:  B         MAIN
```

```
INT0:   B       BADINT
INT1:   B       BADINT
INT2:   B       BADINT
        .space  16 * 16
TINT:   B       BADINT
RINT:   B       BADINT
XINT:   B       BADINT
USER:   B       BADINT

        .bss    ANSWER,2
        .bss    SEED,2

        .bss    PN,2
        .bss    SCLPN,2

        .bss    FC,2
        .bss    FCDEL,2

        .bss    FBASE,2
        .bss    F0,2

        .bss    DOPF,2
        .bss    FLO,2

        .bss    TLO,2
        .bss    THI,2

        .bss    A,1
        .bss    ZERO,1
        .bss    ONE,1
        .bss    C0FFFF,1

        .bss    X0,1
        .bss    X1,1
        .bss    Y0,1
        .bss    Y1,1

        .bss    Z0,1
        .bss    Z1,1
        .bss    Z2,1
        .bss    Z3,1
        .bss    SIGN,1

        .bss    CMD,1
        .bss    DATA1,1
        .bss    DATA2,1

        .bss    WPNT,1
        .bss    STAT,1
        .bss    MODE,1

        .bss    HTAB,8
        .bss    HEND,1
        .bss    LTAB,4
        .bss    LEND,1

        .text

BADINT:
        SXF
BADLOP: B       BADLOP

MAIN:
        DINT                    ; 1     ; Ensure interrupts disabled
```

```
        ROVM                    ; 1     ; Reset overflow mask
        LARP    AR1             ; 1     ; Set ARP to AR0
        LDPK    0               ; 1     ; Point to page 0 (control area)
        LACK    0               ; 1     ; Mask off all interrupts
        SACL    4               ; 1
        RSXM                    ; 1     ; Disable sign extension
        SPM     0               ; 1     ; No shift on P register

        LDPK    PN              ; 1     ; Point to data area

        IN      STAT,STATIN     ; 2     ; Load in status for config info
        CALL    BCDINI                  ; 25    ; Initialize tables for BCD convert
        CALL    RANINI                  ; 19    ; Initialize random number gen
        CALL    SETDEF                  ; 86    ; Set defaults for calculations


        IN      CMD,CMDIN       ; 2     ; Ensure command latch is cleared
        LRLK    AR0,CMD                 ; 3     ; No commands received yet
        SAR     AR0,WPNT
        SXF                     ; 1     ; Latch synth immediately
        CALL    PRETTY
        RXF                     ; 1     ; Latch synth on hop clock

        BIT     STAT,BGT        ; 4     ; Check GT or Payload
        BBZ     ISPAY

ISGT:   LRLK    AR0,MSTOP       ; 3     ; GT config - begin in Mode Stop
        SAR     AR0,MODE
        B       STOP

ISPAY:  LRLK    AR0,MRUN        ; 3     ; Payload config - begin in Mode run
        SAR     AR0,MODE
        B       RUN

STOP:                                   ; Stop Mode - no hopping
        CALL    INSTR                   ; Check if commanded, change modes
        BLZ     STRUN
        BNZ     SYNC
        B       STOP
STRUN:  CALL    RANJMP                          ; Get random # for given time
        B       RUN

RUN:                                    ; Run Mode - normal hopping
        CALL    WAIT                    ; Wait until leading edge of hop
        CALL    LOADPN
        CALL    CALC            ; 352
        OUT     ANSWER,SYNTHL                   ; Output to synth, wait for hop to latch
        OUT     ANSWER+1,SYNTHH
        CALL    RAN             ; Set up Random # for next hop
        CALL    INCTIM
        CALL    INSTR           ; Check if commanded, change modes
        BZ      STOP
        BGZ     SYNC
        B       RUN

SYNC:                                   ; Sync Mode - fast response hops
        CALL    RANJMP                  ; Calc 1st sync hop (T+0)
        CALL    LOADPN
        CALL    CALC
        LARP    AR1
        LRLK    AR1,MG00
        SACL    *+
        SACH    *+
```

```
        CALL    RAN             ; Calc 2nd sync hop (T+2)
        CALL    INCTIM
        CALL    RAN
        CALL    INCTIM
        CALL    LOADPN
        CALL    CALC
        LARP    AR1
        LRLK    AR1,MGO1
        SACL    *+
        SACH    *+

        CALL    RAN             ; Calc 3rd sync hop (T+4)
        CALL    INCTIM
        CALL    RAN
        CALL    INCTIM
        CALL    LOADPN
        CALL    CALC
        LARP    AR1
        LRLK    AR1,MGO2
        SACL    *+
        SACH    *+

        CALL    RAN             ; Calc 4th sync hop (T+6)
        CALL    INCTIM
        CALL    RAN
        CALL    INCTIM
        CALL    LOADPN
        CALL    CALC
        LARP    AR1
        LRLK    AR1,MGO3
        SACL    *+
        SACH    *+


        CALL    STPTIM              ; Calc 5th sync hop (T+5120)
        CALL    RANJMP
        CALL    LOADPN
        CALL    CALC
        LARP    AR1
        LRLK    AR1,MGO4
        SACL    *+
        SACH    *+

        CALL    RAN             ; Set random # and time to hop after
        CALL    INCTIM                  ; 5th sync hop - so can RUN immediately

        SXF                     ; Set to immediate latch mode
        LARP    AR1
        LRLK    AR1,MGO0
        OUT     *+,SYNTHL       ; Output first hop to synth
        OUT     *,SYNTHH

SLOOP:                          ; Wait for command
        BIOZ    SLOOP
        IN      CMD,CMDIN
        ZALH    CMD
        BZ      SSTOP                   ; Respond to Stop Mode command
        BLZ     SRUN            ; Respond to Run Mode command
        LAR     AR1,CMD                 ; Respond to Go commands
        OUT     *+,SYNTHL
        OUT     *,SYNTHH
        B       SLOOP
SSTOP:  RXF                     ; Set to hop clock latch mode
        SACH    MODE
```

```
        B       STOP
SRUN:   RXF                     ; Set to hop clock latch mode
        SACH    MODE
        B       RUN

;*****************************************************************************
;
;                       INSTR - check for and process command
;
;-----------------------------------------------------------

INSTR:
        BIOZ    IDONE                   ; Input latch not full?
        LARP    AR1             ; Get command
        LAR     AR1,WPNT
        IN      *+,CMDIN
        SAR     AR1,WPNT
        LRLK    AR0,DATA1+1     ; Cmd + 1 data ?
        CMPR    0
        BBNZ    IDONE
        LRLK    AR0,DATA2+1     ; Cmd + 2 data ?
        CMPR    0
        BBNZ    LOADS
        LAR     AR1,CMD                 ; Run mode command?
        LRLK    AR0,MRUN
        CMPR    0
        BBNZ    CMODE
        LRLK    AR0,MSYNC       ; Sync mode command?
        CMPR    0
        BBNZ    CMODE
        LRLK    AR0,MSTOP       ; Stop mode command?
        CMPR    0
        BBNZ    CMODE
        LRLK    AR0,MSETDF      ; Set defaults command?
        CMPR    0
        BBNZ    CSETDF
        LRLK    AR0,MRXF        ; Reset XF?
        CMPR    0
        BBNZ    CRXF
        LRLK    AR0,MSXF        ; Set XF?
        CMPR    0
        BBNZ    CSXF
        LRLK    AR0,MGO0        ; Valid go command ?
        CMPR    1               ; (GO 0 to GO 4)
        BBNZ    ERROR
        LRLK    AR0,MGO4+1
        CMPR    1
        BBNZ    CGOS
        LRLK    AR0,MLDANS      ; Valid load command ?
        CMPR    1               ; (LOAD ANS to LOAD THI)
        BBNZ    ERROR
        LRLK    AR0,MLDTHI+1
        CMPR    1
        BBNZ    CLOAD
        LRLK    AR0,MREAD
        CMPR    1
        BBNZ    ERROR
        LRLK    AR0,MRDEND
        CMPR    1
        BBNZ    READS
        B       ERROR                   ; Unrecognized command
```

```
READS:  ZALS    CMD             ; Read values through XF
        ANDK    0FFFH                   ; MSB to LSB
        SACL    CMD             ; 1 bit / command
        LAR     AR1,CMD                 ; STOP to finish
        ZALH    *
        RXF
RDWAIT: BIOZ    RDWAIT
        ROL
        BC      RDSET
RDCLR:  RXF
        B       RDNEXT
RDSET:  SXF
RDNEXT: IN      CMD,CMDIN
        LAR     AR1,CMD
        BANZ    RDWAIT
        RXF
        B       INEXT

LOADS:  LAR     AR1,CMD                 ; Load commands
        ZALS    DATA1
        SACL    *+
        ZALS    DATA2
        SACL    *-
        LRLK    AR0,MLDANS      ; If LOAD answer, output to latches
        CMPR    0
        BBZ     ISLDTH
        OUT     ANSWER,SYNTHL           ; Output to synth, wait for hop to latch
        OUT     ANSWER+1,SYNTHH
        B       INEXT
ISLDTH: LRLK    AR0,MLDTLO      ; If LOAD TLO, zero THI
        CMPR    0
        BBZ     INEXT
        LRLK    AR1,MLDTHI
        SACH    *+
        SACH    *-
        B       INEXT

CMODE:  LAR     AR1,CMD                 ; Change mode commands
        SAR     AR1,MODE        ; Save mode
        B       INEXT

CGOS:   B       INEXT                   ; Go commands, ignored

CLOAD:  B       IDONE                   ; Load commands, held until all data
                                ; is present

CSETDF: CALL    SETDEF                  ; Set default command
        B       INEXT

CRXF:   RXF                     ; Reset XF command
        B       INEXT

CSXF:   SXF                     ; Set XF command
        B       INEXT

ERROR:  B       INEXT                   ; Errors, unrecognized command

INEXT:  LRLK    AR1,CMD                 ; Reset input pointer to accept the
        SAR     AR1,WPNT        ; next command
        B       IDONE

IDONE:  ZALH    MODE                    ; Load the next (or current) mode
        RET                     ; and return
```

```
;*****************************************************************
;
;                    WAIT - wait for upswing of hop clock
;
;---------------------------------------------------------------

WAIT:
        IN      STAT,STATIN         ; Ensure clock is low
        ZALH    STAT
        BLZ     WAIT
WAITUP:
        IN      STAT,STATIN         ; Wait for low to high transition
        ZALH    STAT
        BGZ     WAITUP
        RET

;*****************************************************************
;
;                    INCTIM - Advance time by one hop
;
;---------------------------------------------------------------

INCTIM:
        LARP    AR1
        LAR     AR1,TLO
        ADRK    1
        SAR     AR1,TLO
        LRLK    AR0,61440
        CMPR    0
        BBNZ    INCDON                  ; If at max, reset seed and time
        LARK    AR1,1               ; This allows a periodic random
        SAR     AR1,SEED            ;  sequence with period 61440 hops
        LARK    AR1,0
        SAR     AR1,SEED+1
        SAR     AR1,TLO
        SAR     AR1,THI
INCDON: RET

;*****************************************************************
;
;                    STPTIM - Advance time by 16 frames for next sync burst
;
;---------------------------------------------------------------

STPTIM:
        ZALS    TLO                 ; Note that time has already been
        ADLK    5120-6              ;  advanced by 6 hops.
        SACL    TLO
        RET

;*****************************************************************
;
;                    LOADPN - load a PN value from the seed
;
;---------------------------------------------------------------

LOADPN:
        ZALS    SEED
```

```
        SACL    PN
        ZALS    SEED+1
        ANDK    00FFH                   ; Mask off upper 8 bits giving 24 bits
        SACL    PN+1
        RET

;*****************************************************************************
;
;       SETDEF - sets default starting values for calculation parameters
;
;-----------------------------------------------------------------------------
;
;               in:     <nil>
;               out:    PN,SCLPN,FC,FCDEL,FBASE,F0,DOPF,FLO,TLO,THI
;               used:   <nil>
;
;               cycles:
;                       86              (includes call/return)
;               time:
;                       8.6us           (includes call/return)
;
;-----------------------------------------------------------------------------

        .global SETDEF
        .global PN,SCLPN,FC,FCDEL,FBASE,F0,DOPF,FLO,TLO,THI

        .data
                        ; GT Uplink
GUPN:   .long   0               ; PN sequence 24 bits 0-16777215
                                ;  max value 16777215
GUSCLP: .long   9990000                 ; PN Scale (x2^23) = 1.19090080261
                                ;  gives scaled PN of 0-19979998 [100Hz]
                                ;  approx = 1.998GHz
                                ;  max value is 499500000 approx 59.4
                                ;  corresponding to 99.9GHz
GUFC:   .long   0               ; FSK(3LSB 0-7)/Channel(5MSB 0-31)
                                ;  max composite value is 255
GUFCDL: .long   400             ; Tone spacing 400 [100Hz] = 40kHz
                                ;  max value is 3.2768MHz
GUFBAS: .long   435000000       ; Base freq of 2GHz range [100Hz] = 43.5GHz
                                ;  max value is 99.9GHz
GUF0:   .long   5000            ; Guard offset from base [100Hz] = 500kHz
                                ;  max value is 99.9GHz
GUDOPF: .long   0               ; Doppler factor (x2^32)
                                ;  max value is +42950 = +1.00000761449E-05
                                ;    gives 400kHz on 40GHz
                                ;  smallest delta is 1 =  2.32830643654E-10
                                ;    gives 9Hz on 40GHz
                                ;    (note best resolution is 100Hz)
                                ;  min value is -42950 = -1.00000761449E-05
GUFLO:  .long   430000000       ; Upconverter LO [100Hz] = 43GHz
                                ;  this value will not be sent to synth
                                ;  so it is subtracted off before BCD

                        ; GT Downlink
GDPN:   .long   0               ; PN sequence
GDSCLP: .long   4977500                 ; PN Scale (x2^23)=0.5557899475098 (0-9359999)
GDFC:   .long   0               ; FSK(3LSB 0-7)/Channel(5MSB 0-31)
GDFCDL: .long   0               ; Tone spacing 0  (no modulation)
GDFBAS: .long   202000000       ; Base freq of 2GHz range [100Hz] = 20.2GHz
GDF0:   .long   2500            ; Guard offset from base [100Hz] = 250kHz
GDDOPF: .long   0               ; Doppler factor (x2^32)
GDFLO:  .long   192000000       ; Upconverter LO [100Hz] = 19.2GHz
```

```
                        ; Payload Uplink
PUPN:     .long   0             ; PN sequence 24 bits 0-16777215
PUSCLP:   .long   9990000               ; PN Scale (x2^23) = 1.19090080261 (0-19979998)
PUFC:     .long   0             ; FSK(3LSB 0-7)/Channel(5MSB 0-31)
PUFCDL:   .long   0             ; Tone spacing 0 (no modulation)
PUFBAS:   .long   435000000     ; Base freq of 2GHz range [100Hz] = 43.5GHz
PUF0:     .long   5000          ; Guard offset from base [100Hz] = 500kHz
PUDOPF:   .long   0             ; Doppler factor (x2^32)
PUFLO:    .long   430000000     ; Upconverter LO [100Hz] = 43GHz

                        ; Payload Downlink
PDPN:     .long   0             ; PN sequence
PDSCLP:   .long   4977500               ; PN Scale (x2^23)=0.5557899475098 (0-9359999)
PDFC:     .long   0             ; FSK(3LSB 0-7)/Channel(5MSB 0-31)
PDFCDL:   .long   0             ; Tone spacing 0  (no modulation)
PDFBAS:   .long   202000000     ; Base freq of 2GHz range [100Hz] = 20.2GHz
PDF0:     .long   2500          ; Guard offset from base [100Hz] = 250kHz
PDDOPF:   .long   0             ; Doppler factor (x2^32)
PDFLO:    .long   192000000     ; Upconverter LO [100Hz] = 19.2GHz

ALLTLO:   .long   0             ; Time (low longword) in number of hops since
ALLTHI:   .long   0             ;      (high longword) time 0.


          .text
SETDEF:
          BIT     STAT,BGT
          BBZ     PAYLOAD
          BIT     STAT,BUP
          BBZ     GTDOWN
          B       GTUP
PAYLOAD:
          BIT     STAT,BUP
          BBZ     PAYDOW
          B       PAYUP

GTUP:
          BLKP    GUPN,PN                 ; 68      ; Load GT Uplink table into RAM
          BLKP    GUPN+1,PN+1
          BLKP    GUSCLP,SCLPN
          BLKP    GUSCLP+1,SCLPN+1
          BLKP    GUFC,FC
          BLKP    GUFC+1,FC+1
          BLKP    GUFCDL,FCDEL
          BLKP    GUFCDL+1,FCDEL+1
          BLKP    GUFBAS,FBASE
          BLKP    GUFBAS+1,FBASE+1
          BLKP    GUF0,F0
          BLKP    GUF0+1,F0+1
          BLKP    GUDOPF,DOPF
          BLKP    GUDOPF+1,DOPF+1
          BLKP    GUFLO,FLO
          BLKP    GUFLO+1,FLO+1
          B       COMMON

GTDOWN:
          BLKP    GDPN,PN                 ; 68      ; Load GT Downlink table into RAM
          BLKP    GDPN+1,PN+1
          BLKP    GDSCLP,SCLPN
          BLKP    GDSCLP+1,SCLPN+1
          BLKP    GDFC,FC
          BLKP    GDFC+1,FC+1
          BLKP    GDFCDL,FCDEL
          BLKP    GDFCDL+1,FCDEL+1
```

```
        BLKP    GDFBAS,FBASE
        BLKP    GDFBAS+1,FBASE+1
        BLKP    GDF0,F0
        BLKP    GDF0+1,F0+1
        BLKP    GDDOPF,DOPF
        BLKP    GDDOPF+1,DOPF+1
        BLKP    GDFLO,FLO
        BLKP    GDFLO+1,FLO+1
        B       COMMON

PAYUP:
        BLKP    PUPN,PN                 ; 68    ; Load Payload Uplink table into RAM
        BLKP    PUPN+1,PN+1
        BLKP    PUSCLP,SCLPN
        BLKP    PUSCLP+1,SCLPN+1
        BLKP    PUFC,FC
        BLKP    PUFC+1,FC+1
        BLKP    PUFCDL,FCDEL
        BLKP    PUFCDL+1,FCDEL+1
        BLKP    PUFBAS,FBASE
        BLKP    PUFBAS+1,FBASE+1
        BLKP    PUF0,F0
        BLKP    PUF0+1,F0+1
        BLKP    PUDOPF,DOPF
        BLKP    PUDOPF+1,DOPF+1
        BLKP    PUFLO,FLO
        BLKP    PUFLO+1,FLO+1
        B       COMMON

PAYDOW:
        BLKP    PDPN,PN                 ; 68    ; Load Payload Downlink table into RAM
        BLKP    PDPN+1,PN+1
        BLKP    PDSCLP,SCLPN
        BLKP    PDSCLP+1,SCLPN+1
        BLKP    PDFC,FC
        BLKP    PDFC+1,FC+1
        BLKP    PDFCDL,FCDEL
        BLKP    PDFCDL+1,FCDEL+1
        BLKP    PDFBAS,FBASE
        BLKP    PDFBAS+1,FBASE+1
        BLKP    PDF0,F0
        BLKP    PDF0+1,F0+1
        BLKP    PDDOPF,DOPF
        BLKP    PDDOPF+1,DOPF+1
        BLKP    PDFLO,FLO
        BLKP    PDFLO+1,FLO+1
        B       COMMON

COMMON:
        BLKP    ALLTLO,TLO
        BLKP    ALLTLO+1,TLO+1
        BLKP    ALLTHI,THI
        BLKP    ALLTHI+1,THI+1
        RET


PRETTY:
        ZAC
        SACL    ANSWER
        LARP    AR1
        LARK    AR1,9
PLOOP:
        OUT     ANSWER,SYNTHL
```

```
        OUT     ANSWER,SYNTHH
        LALK    20000
        SACL    ANSWER+1
POSLP:                          ; } Outer loop
        RPTK    255             ; }                     } Inner loop
        ZALS    ANSWER+1        ; }                     }
        SUBK    1               ; }
        SACL    ANSWER+1        ; }
        BNZ     POSLP                   ; }

        LALK    CPRETY
        ADD     ANSWER
        SACL    ANSWER
        BANZ    PLOOP                   ; Test, then decrement AR1
        OUT     ZERO,SYNTHL
        OUT     ZERO,SYNTHH
        RET

        RET
        .end
```

3. Calculation Subroutines (CALC.ASM)

```
        .length   55
        .width    120
        .global   CALC,MUL32,BCD32,BCDINI
        .global ANSWER,SEED,A,ZERO,ONE,C0FFFF
        .global PN,SCLPN,FC,FCDEL,FBASE,F0,DOPF,FLO,TLO,THI
        .global X0,X1,Y0,Y1,Z0,Z1,Z2,Z3,SIGN
        .global   CMD,DATA1,DATA2,WPNT,STAT,MODE
        .global HTAB,HEND,LTAB,LEND

BGT:    .set    7
BUP:    .set    6
BINTCK: .set    5
BCOMST: .set    4
BIGHZ:  .set    3
BBIO:   .set    2
BSPARE: .set    1
BHOP:   .set    0


;=================================================================================
;
;       CALC - calculates synth output value from PN input
;
;---------------------------------------------------------------------------------
;
;               in:
;                       PN+1,PN - 24 bit PN sequence input
;                       FC - FSK tone (3 lsb) and channel number (5 msb)
;                       SCLPN,FCDEL,FBASE,F0,DOPF,FLO - as describes in MAIN
;               out:
;                       ANSWER and ACC - 32 bit result
;               used:
;                       ARP,AR0,AR1,ACC,T,P
;
;               cycles:
;                       352             (includes call/return)
;               time:
;                       35.2us          (includes call/return)
;
;---------------------------------------------------------------------------------

        .text
CALC:
        ZALS    PN              ; 69    ; Scale PN sequence
        SACL    X0
        ZALS    PN+1
        SACL    X1
        ZALS    SCLPN
        SACL    Y0
        ZALS    SCLPN+1
        SACL    Y1
        CALL    MUL32           ; 54
        LAC     Z1,9
        SACH    X0
        LAC     Z3,9
        SACL    X1
        ZALH    X1
        ADD     Z2,9
        ADD     X0

        LT      FC              ; 3     ; Calculate FSK and channel freq
```

```
        MPY     FCDEL                           ; and add to PN freq
        APAC

        ADDS    FBASE           ; 2     ; Add in base freq
        ADDH    FBASE+1

        ADDS    F0      ; 2     ; Add in offset from base
        ADDH    F0+1

        SACL    X0      ; 64    ; Compute dopple and add in
        SACH    X1
        ZALS    DOPF
        SACL    Y0
        ZALS    DOPF+1
        SACL    Y1
        CALL    MUL32   ;54
        ZALH    X1
        ADDS    X0
        ADDS    Z2
        ADDH    Z3

        SUBS    FLO     ; 2     ; Subtract off the uplink LO freq
        SUBH    FLO+1

        BIT     STAT,BCOMST ; 4 ; Check if Comstron or Scitek
        BBZ     SCITEK
COMSTR:                 ; 202   ; Convert to BCD and store
        SACL    X0
        SACH    X1
        CALL    BCD32   ;198
        ZALH    Z1
        ADDS    Z0
        BIT     STAT,B1GHZ  ; 14    ; If 1GHz offset is set XOR 1GHz bit
        BBZ     FINISH
        XORK    2000H,15
        B       FINISH
SCITEK:                 ; 5     ; Scitek not implemented yet
        ZAC

FINISH:
        SACL    ANSWER          ; 2
        SACH    ANSWER+1
        RET

;=================================================================
;
;       BCD32 - binary to BCD conversion 32 bit output (8 digits)
;
;-----------------------------------------------------------------
;
;               in:
;                       X1,X0 - binary value, 0-9999999 (5F5E0FFH)
;               out:
;                       Z1,Z0 - BCD value
;               used:
;                       ARP,AR0,AR1,ACC
;
;               cycles:
;                       198             (includes call/return)
;               time:
;                       19.8us          (includes call/return)
;
;-----------------------------------------------------------------
```

```
        .text
BCD32:
        LARP    1               ; 1     ; Point to BCD table
        LRLK    AR1,HTAB        ; 2
        LRLK    AR0,HEND        ; 2
        ZALH    X1              ; 1     ; Load in value to be converted
        ADDS    X0              ; 1 [7]

LOOP:
        SUBS    *+              ; 1(9)  ; Determine if 8 position is required
        SUBH    *-              ; 1     ;  by subtracting and checking borrow.
        BC      DID8            ; 3     ;  All operations 32 bits
        ADDS    *+              ; 1     ; Not required, so add constant back
        ADDH    *-              ; 1
        RC                      ; 1     ; Clear carry for 0 in 8 position
DID8:   ROL                     ; 1     ; Double ACC and add carry

        SUBS    *+              ; 1(9)  ; Determine if 4 position is required
        SUBH    *-              ; 1
        BC      DID4            ; 3
        ADDS    *+              ; 1
        ADDH    *-              ; 1
        RC                      ; 1
DID4:   ROL                     ; 1

        SUBS    *+              ; 1(9)  ; Determine if 2 position is required
        SUBH    *-              ; 1
        BC      DID2            ; 3
        ADDS    *+              ; 1
        ADDH    *-              ; 1
        RC                      ; 1
DID2:   ROL                     ; 1

        SUBS    *+              ; 1(9)  ; Determine if 1 position is required
        SUBH    *-              ; 1
        BC      DID1            ; 3
        ADDS    *+              ; 1
        ADDH    *-              ; 1
        RC                      ; 1
DID1:   ROL                     ; 1

        ADRK    2               ; 1(5) 1 loop = 41, 4 loops = 164
        CMPR    0               ; 1
        BBZ     LOOP            ; 3 [171]  ; Do for high 16 bits

        SACL    Z1              ; 1(5)          ; Do the low 16 bit conversion
        SACH    Z0              ; 1
        LAC     Z0,15           ; 1
        LRLK    AR1,LTAB        ; 2 [176]

        SUBC    *               ; 1(16) ; Conditional subtraction, similar to
        SUBC    *               ; 1     ;  above, but only 16 bits.
        SUBC    *               ; 1
        SUBC    *+              ; 1
        SUBC    *               ; 1
        SUBC    *               ; 1
        SUBC    *               ; 1
        SUBC    *+              ; 1
        SUBC    *               ; 1
        SUBC    *               ; 1
        SUBC    *               ; 1
        SUBC    *+              ; 1
        SUBC    *               ; 1
        SUBC    *               ; 1
```

```
        SUBC    *               ; 1
        SUBC    *+              ; 1
        SACL    Z0              ; 1 [192]  ; Store lower 16 bits

        RET             ; 3ret+3call[198] 198 cycles @ 100ns = 19.8 us

;***************************************************************************
;
;       BCDINI - initialization for binary to BCD conversion (BCD32)
;
;---------------------------------------------------------------------------
;
;               in:     <nil>
;               out:
;                       HTAB - table of long constants (8 words)
;                       LTAB - table of short constants (4 words)
;               used:
;                       ARP,AR1
;
;               cycles:
;                       25              (includes call/return)
;               time:
;                       2.5us           (includes call/return)
;
;---------------------------------------------------------------------------

        .data

HTABLE: .long   80000000                ; Constant table for 32 bit operations
        .long   8000000*16              ; to compute high 16 bits.
        .long   800000*16*16
        .long   80000*16*16*16

LTABLE: .word   8000                    ; Constant table for 16 bit operations
        .word   800*16                          ; to compute low 16 bits
        .word   80*16*16
        .word   8*16*16*16

        .text
BCDINI:
        LARP    1               ; 1     ; Load high table into RAM
        LRLK    AR1,HTAB        ; 2
        RPTK    7               ; 1
        BLKP    HTABLE,*+       ; 11
        LRLK    AR1,LTAB        ; 2     ; Load low table into RAM
        RPTK    3               ; 1
        BLKP    LTABLE,*+       ; 7 [25]

        RET             ; 3ret+3call[25] 25 cycles @ 100ns = 2.5 us

;***************************************************************************
;
;       MUL32 - 32 x 32 = 64 bit signed multiply
;
;---------------------------------------------------------------------------
;
;               in:
;                       X1,X0 - value1
;                       Y1,Y0 - value2
;               out:
;                       Z3,Z2,Z1,Z0 - result
;               used:
```

```
;                       ACC,T,P
;
;               cycles:
;                       54              (includes call/return)
;               time:
;                       5.4us           (includes call/return)
;
;-------------------------------------------------------

        .text
MUL32:
        ZALS    X1      ; 1(3)  ; Determine sign of product
        XOR     Y1      ; 1
        SACH    SIGN,1          ; 1[3]

ABSX:   ZALH    X1      ; 1(5)  ; Take absolute value of X
        ADDS    X0      ; 1
        ABS             ; 1
        SACH    X1      ; 1
        SACL    X0      ; 1[8]

ABSY:   ZALH    Y1      ; 1(5)  ; Take absolute value of Y
        ADDS    Y0      ; 1
        ABS             ; 1
        SACH    Y1      ; 1
        SACL    Y0      ; 1[13]

MULT:   LT      X0      ; 1(4)  ; X0*Y0 -> Z1,Z0
        MPYU    Y0      ; 1
        SPL     Z0      ; 1
        SPH     Z1      ; 1[17]

        MPYU    Y1      ; 1(2)  ; X0*Y1 -> acc
        LTP     X1      ; 1[19]

        MPYU    Y0      ; 1(2)  ; X0*Y1 + (X0*Y0)H -> acc
        ADDS    Z1      ; 1[21]

        MPYA    Y1      ; 1(11) ; X1*Y0 + X0*Y1 + (X0*Y0) -> Z2,Z1
        SACL    Z1      ; 1
        SACH    Z2      ; 1
        ZALS    Z2      ; 1     ; X1*Y1 + Z2 + carry -> Z3,Z2
        BNC     SUM     ; 3
        ADDH    ONE     ; 1
SUM:    APAC            ; 1
        SACL    Z2      ; 1
        SACH    Z3      ; 1[32]

        LAC     SIGN    ; 1(4)  ; Fix sign
        BZ      DONE    ; 3[36]

        ZALH    Z1      ; 1(12) ;Negate product
        ADDS    Z0      ; 1
        CMPL            ; 1
        ADD     ONE     ; 1
        SACL    Z0      ; 1
        SACH    Z1      ; 1
        ZALS    Z2      ; 1
        ADDH    Z3      ; 1
        CMPL            ; 1
        ADDC    ONE     ; 1
        SACL    Z2      ; 1
        SACH    Z3      ; 1[48]
```

```
DONE:   RET             ; 3ret+3call[54] 54 cycles @ 100ns = 5.4 us
        .end
```

4. Random Number Generation Routines (RAN.ASM)

```
        .length     55
        .width      120
        .global RAN,RANINI,RANJMP
        .global SEED,A,ZERO,ONE,C0FFFF
        .global PN,SCLPN,FC,FCDEL,FBASE,F0,DOPF,FLO,TLO,THI
        .global X0,X1,Y0,Y1,Z0,Z1,Z2,Z3,SIGN

;*****************************************************************************
;
;       RAN - computes next random number
;
;-----------------------------------------------------------
;
;               in:     <nil>
;               out:
;                       SEED+1,SEED and ACC
;               used:
;                       ACC,P,T
;
;               cycles:
;                       24              (includes call/return)
;               time:
;                       2.4us           (includes call/return)
;
;-----------------------------------------------------------

        .text

RAN:
        LT      A               ; Multiply low seed by constant
        MPYU    SEED
        SPL     Z0
        SPH     Z1

        ZALS    Z1              ; Load in partial sum /2^16
        MPYU    SEED+1                  ; Multiply high seed by constant
        APAC                    ; Add in
        ROL                     ; Shift one bit left and mask off all but
        SACH    Z2              ;  lower 31 bits then shift back
        AND     C0FFFF
        ROR
        SACL    Z3
        ZALH    Z3              ; Load in partial sum
        ADDS    Z0              ; Add in from original low multiply
        ADDS    Z2              ; Add in high word (see modified method in
        ADDC    ZERO            ;  the paper) and add in carry 9overflow)

        SACL    SEED            ; Save new seed
        SACH    SEED+1
        RET


;*****************************************************************************
;
;       RANINI - intializes values for pseudo random generation
;
;-----------------------------------------------------------
;
;               in:     <nil>
```

```
;                   out:
;                         ZERO,ONE,C0FFFF,SEED,A
;                   used:
;                         ACC
;
;                   cycles:
;                         19                (includes call/return)
;                   time:
;                         1.9us             (includes call/return)
;
;-------------------------------------------------------

RANINI:
          LACK    0
          SACL    ZERO
          LACK    1
          SACL    ONE
          LALK    0FFFFH
          SACL    C0FFFF

          LALK    16807          ; A = 16807
          SACL    A

          LACK    1              ; Seed <- 00000001
          SACL    SEED
          SACH    SEED+1
          RET


;=======================================================================
;
;         RANJMP - jump to certain place in random sequence
;
;-------------------------------------------------------
;
;                   in:
;                         TLO - number to jump to
;                   out:
;                         SEED+1,SEED and ACC
;                   used:
;                         ARP,AR0,AR1,ACC,T,P
;
;                   cycles:
;                         13+29/jmp                 (includes call/return)
;                   time:
;                         1.3us + 2.9us/jmp   (includes call/return)
;
;                         for 10000 jumps, time is 29ms
;
;-------------------------------------------------------

RANJMP:
          LARP    AR1
          LARK    AR1,1
          SAR     AR1,SEED
          LARK    AR1,0
          SAR     AR1,SEED+1
          LAR     AR0,TLO
          CMPR    0
          BBNZ    DONE
LOOP:
          CALL    RAN
          ADRK    1
```

```
        CMPR    0
        BBZ     LOOP
DONE:
        RET

        .end
```

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1. ORIGINATOR (the name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Establishment sponsoring a contractor's report, or tasking agency, are entered in section 8.)

Defence Research Establishment Ottawa
Ottawa, Ontario
K1A 0Z4

2. SECURITY CLASSIFICATION (overall security classification of the document. including special warning terms if applicable)

UNCLASSIFIED

3. TITLE (the complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S,C or U) in parentheses after the title.)

Hopping Synthesizer Controller (U)

4. AUTHORS (Last name, first name, middle initial)

Addison, Robin D.

5. DATE OF PUBLICATION (month and year of publication of document)

August 1990

6a. NO. OF PAGES (total containing information. Include Annexes, Appendices, etc.)

80

6b. NO. OF REFS (total cited in document)

4

7. DESCRIPTIVE NOTES (the category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

Technical Note

8. SPONSORING ACTIVITY (the name of the department project office or laboratory sponsoring the research and development. Include the address.)

Defence Research Establishment Ottawa
Electronics Division/SATCOM Section
Ottawa, Ontario K1A 0Z4

9a. PROJECT OR GRANT NO. (if appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant)

Project D6470

9b. CONTRACT NO. (if appropriate, the applicable number under which the document was written)

10a. ORIGINATOR'S DOCUMENT NUMBER (the official document number by which the document is identified by the originating activity. This number must be unique to this document.)

DREO Technical Note 90-17

10b. OTHER DOCUMENT NOS. (Any other numbers which may be assigned this document either by the originator or by the sponsor)

11. DOCUMENT AVAILABILITY (any limitations on further dissemination of the document, other than those imposed by security classification)

(X) Unlimited distribution
( ) Distribution limited to defence departments and defence contractors; further distribution only as approved
( ) Distribution limited to defence departments and Canadian defence contractors; further distribution only as approved
( ) Distribution limited to government departments and agencies; further distribution only as approved
( ) Distribution limited to defence departments; further distribution only as approved
( ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availabilty (11). However, where further distribution (beyond the audience specified in 11) is possible, a wider announcement audience may be selected.)

Unlimited Announcement

UNCLASSIFIED

SECURITY CLASSIFICATION OF FORM

13. ABSTRACT (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), or (U). It is not necessary to include here abstracts in both offical languages unless the text is bilingual).

The EHF SATCOM (Extremely High Frequency Satellite Communications) project includes a contract in industry for a ground-based model of a frequency hopping spread spectrum satellite communications system. There is also an in-house effort in key technical areas. One requirement is a Hopping Synthesizer Controller to drive a frequency synthesizer. The controller was built on a single board using a digital signal processor for calculations and an erasable programmable logic device for interfacing. It provides the pseudo-random hopping sequence, modulation, channel selection and doppler precorrection. The Hopping Synthesizer Controller can drive the frequency synthesizer at hop rates up to 25 khops/s. Four configurations are supported: operation within the payload or ground terminal simulator for uplind and downlink. Each configuration supports commands from a Synchronization Controller and a special synchronization mode. In this document the software and hardware details are provided along with a user guide for the board.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

Spread Spectrum
Frequency Hopping
Synthesizer Controller
Military Communications
Satellite Communications